IBM

Reference Manual

IBM 7070/7074 Four-Tape Autocoder

IBM

Reference Manual

IBM 7070/7074 Four-Tape Autocoder

REVISED APRIL 1961

The following options have been added for Four-Tape Autocoder operation:

1. Stacking
2. Messages
3. Tape density
4. Choice of parameters in skeleton routines

The following Input/Output Control System sections have been added:

1. Restrictions on the use of the Input/Output Control System with Four-Tape Autocoder.
2. A list of halts and messages that may occur during assembly.

Part III has been revised.

A detailed list of changes incorporated into this manual is given in Appendix F.

This edition supersedes the previous manual, form C28-6102.

Contents

INTRODUCTION

PART I—7070/7074 FOUR-TAPE AUTOCODER LANGUAGE 1
7070/7074 Four-Tape Autocoder Statements 1
Coding Sheet 1
Address Types 3
Field Definition 4
Address Adjustment 5
Index Words 5
Electronic Switches 7
Input/Output Unit and Alteration Switch Designations 7
Remarks and Comments 7
Imperative Operations 8
Declarative Operations 9
DA—Define Area 10
DA Header Line 10
Succeeding Entries 12
Relative Field Definition 14
DC—Define Constant 15
DC Header Linc 15
Numerical Constants and Adcons 16
Alphameric Constants 18
DRDW—Define Rccord Definition Word 20
EQU—Equate 21
Actual or Symbolic Address 21
Index Word or Electronic Switch Number 22
Channel, Channel and Unit, Arm and File, Unit Record and Inquiry Synchronizers, Alteration Switches 23
Control Operations 23
Origin Control 23
Litorigin Control 25
Branch Control 26
End Control 27

PART II—7070/7074 FOUR-TAPE AUTOCODER LIBRARY 29
Macro-Instructions 29
ZSUM 30
CALL 31
Writing Substitution-type Macro-instructions 31
Subroutines 37
Writing Subroutines 38

PART III—7070/7074 FOUR-TAPE AUTOCODER PROCESSOR 41
Organization of the Processor 41
Program Functioning 41
System Control 41
Librarian 41
Phase 1 41
Phase 2 43
Phase 3 44
Phase 4 44

Types of Runs 45
Original Assembly Run 45
Re-assembly Run 45
System Run 46
RUN Control Cards 48
Option Cards 49
Stacking Input and Output 53
Stacking Source Programs 53
Stacking Edited Listings 54
Stacking Object Programs 54
Operating Instructions 54
Preparations for an Original Assembly 55
Preparations for a Re-assembly 56
Preparations for a System Run 56
Console Procedure 57
Halts and Messages 58
Messages 61
System Control and Librarian 61
Phase 1 62
Phase 2 63
Phase 3 64
Phase 4 65
Input/Output Control System 66

Appendix A—Processor Flow Charts 69

Appendix B—Alphabetic List of 7070 Autocoder Imperative Operation Codes 74

Appendix C—Glossary 78

Appendix D—Note on Optional Characters 80

Appendix E—Use of the Input/Output Control System 81

Appendix F—Changes to the Four-Tape Autocoder Manual 82

Index 85

Introduction

The 7070/7074 Four-Tape Autocoder is a symbolic programming system designed to simplify the preparation of programs for the 7070 and 7074 Data Processing Systems. With the increased capacity and versatility of data processing systems, machine-language instructions have increased correspondingly in both number and complexity. The fact that coding in machine language today is an extremely tedious and time-consuming task has led to the development of symbolic programming systems like 7070/7074 Four-Tape Autocoder. These systems permit the programmer to code in symbolic languages which are more meaningful and easier to handle than numerical machine language. They also perform automatically many burdensome tasks such as assigning and keeping track of storage locations and checking for errors. Use of these systems will save the programmer a significant amount of valuable programming time and effort.

This manual is to serve as a reference text on the 7070/7074 Four-Tape Autocoder. The information presented will enable the reader to code and process his program using the Four-Tape Autocoder system. The manual assumes that the programmer is familiar with the methods of data handling and the functions of instructions in the 7070 Data Processing System; this information is included in the IBM Reference Manual "7070 Data Processing System," form A22-7003-2.

The 7070/7074 Four-Tape Autocoder system is designed for use in installations having a 7070 or 7074 Data Processing System with four or five IBM 729 Magnetic Tape Units. Four-Tape Autocoder provides programming features that are not available to users of 7070 Basic Autocoder. These features include the processing of Library subroutines and "substitution-type" macro-instructions, and a considerable expansion of the functions of the LITORIGIN CNTRL entry. Also, columns 60-75 of the Autocoder Coding Sheet may be used for source program entries.

To avoid the need for reprogramming when advancing from Basic Autocoder to Four-Tape Autocoder, 7070/7074 Four-Tape Autocoder has been designed to process any program that can be assembled with 7070 Basic Autocoder. The reverse, however, is not necessarily true.

The ability to process macro-instructions can simplify programming and reduce the time required to write a program. Although the method of writing "substitution-type" macro-instructions for processing by Four-Tape Autocoder differs from, and is less powerful than, the macro-generator method available in the more advanced IBM 7070/7074 Autocoder system (requiring six tape units), it will be found to be an extremely valuable and time-saving programming feature. If the user advances from Four-Tape Autocoder to IBM 7070/7074 Autocoder at some future date, the "substitution-type" macro-instructions may be duplicated by the use of macro generators or the program may be recoded to utilize macro-instructions provided with, or added by the user to, the Autocoder system.

The subroutines and "substitution-type" macro-instructions processed by Four-Tape Autocoder may come from three sources as follows:

1. Macro-instructions which are part of the Input/Output Control System may be used as explained in Appendix E; these macros are described in the 7070 Data Processing System Bulletin "IBM 7070 Input/Output Control System," form J28-6033-1.
2. A few macro-instructions are supplied in the Library portion of the Four-Tape Autocoder System Tape.
3. Subroutines and macro-instructions written by the user and inserted into the Library may be utilized.

The Four-Tape Autocoder system consists of three parts: the Autocoder symbolic language used in writing a program, the Library which contains subroutines and macro-instructions that may be incorporated into a program, and the Four-Tape Autocoder processor which translates the program written in Autocoder language to a 7070 machine-language program.

An Autocoder instruction consists of three parts: a label (optional); an operation code; and an operand, or address portion. The Autocoder imperative operation codes are easily remembered alphameric mnemonics ranging from one to five characters; e.g., the Zero Accumulator 1 and Add, Store Accumulator 2, and Halt and Proceed instructions have Autocoder mnemonics of ZA1, ST2 and HP, respectively. Each 7070 machine-language command has a unique Autocoder mnemonic representation, even though the machine-language codes may be the same. For example, the Unit Record Read and Unit Record Punch machine commands are both +69; the corresponding Autocoder mnemonics are UR and UP, respectively.

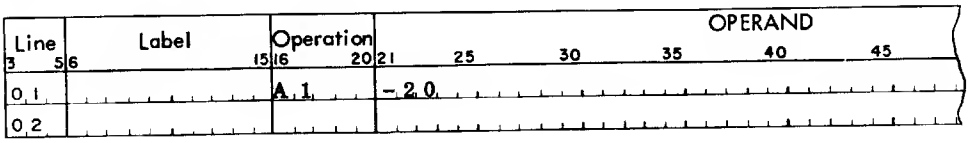
A major advantage of coding in Autocoder language is that the programmer need not be concerned with the actual core storage locations of instructions or data. Instead, he may assign meaningful mnemonic labels, or names, to instructions and to data fields within an area. The programmer may then refer to a certain instruction or operate on a particular data field merely by writing the name which he has assigned to the instruction or data field in the operand of an imperative operation.

In addition to the imperative operation codes, which correspond to the 7070 machine-language codes, the Autocoder language includes "declarative" operations which serve to provide the processor with the information necessary to properly complete the program imperative instructions. They are used by the programmer to name and define the positions and lengths of data fields within an area; to name and enter program constants; to instruct the processor to generate record definition words associated with particular data areas; and, for programming convenience, to give names to items such as alteration switches, unit record synchronizers and index words.

To illustrate Autocoder coding, a programmer might assign, through the DA (Define Area) declarative operation, the following mnemonic names to data fields within a payroll record: HRSWORKED, RATE, EARNINGS. Also, the name CALVACDUE might be given to the first instruction of a routine within a payroll program which is used to calculate the amount of vacation time due an employee. Suppose it is desired to multiply the contents of the field named HRSWORKED by the contents of the field named RATE, store the result in the EARNINGS field, and then branch to the CALVACDUE routine. The Autocoder coding for this operation might appear as follows:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
01		ZA 3	HRSWORKED
02		M	RATE
03		ST 2	EARNINGS
04		B	CALVACDUE
05			
06			
07	CALVACDUE	(First Instruction of vacation-time routine)	
08			

The Autocoder language allows the use of "literals," that is, the actual data to be operated on by an instruction. Thus, if the programmer desires to add the number —20 to the contents of accumulator 1, he may simply enter the symbolic instruction



Besides imperative and declarative operations, the Autocoder language includes "control" operations. These may be used by the programmer to control the placement of data and instructions in storage, and to instruct the processor to generate unconditional Branch instructions to be executed during or following the loading of the "processed" machine-language program.

The program written in the Autocoder language, which is the input to the processor, is called the "source program." The processor will convert the source program mnemonic imperative operation codes into the corresponding machine-language codes and assign core storage addresses to the symbolic instructions, literals and symbolic data references. The assigned core storage addresses, together with the proper field definition, will be inserted by the processor into the corresponding assembled machine-language instructions. Besides performing a number of routine and clerical tasks for the programmer, the processor will also check for common coding errors and indicate these by appropriate messages on the console typewriter.

The use of symbolic names makes a program independent of actual machine locations; therefore, programs and routines written in Autocoder language can be relocated and combined as desired. Routines within a program can be written independently with no loss of efficiency in the final program. Also, symbolic instructions may be simply added or deleted without the necessity of reassigning storage addresses.

The Library contains a few macro-instructions supplied with Four-Tape Autocoder plus subroutines and/or additional macro-instructions written by the user. Use of items from the Library simplify and speed up the writing of programs by providing frequently used routines that may be easily incorporated into programs to be assembled using Four-Tape Autocoder.

The Four-Tape Autocoder processor is divided into six sections as follows:

Systems Control. This section performs "housekeeping" operations for later sections of the processor.

Librarian. Subroutines and macro-instructions may be added to, or deleted from, the Library through the use of this section of the processor.

Phase 1. In this section, subroutines and macro-instructions are taken from the Library and inserted into the program.

Phase 2. Locations are assigned to data and instructions in this part of the processor.

Phase 3. Storage locations are inserted into the operands of instructions to complete the machine-language program.

Phase 4. This section prepares a listing of the program, a program tape and an optional program deck.

The Four-Tape Autocoder processor is used for three types of machine "runs" which are: an original assembly which uses the original source program, a re-assembly which uses the listing tape of a previous assembly plus any additions, deletions and corrections, and a system run which reproduces and/or changes the contents of the Four-Tape Autocoder System Tape.

The 7070/7074 Four-Tape Autocoder processor can assemble programs for use with *any* configuration of 7070 or 7074 system. However, the processor itself requires only a Data Processing System having 5,000 words of core storage and four magnetic tape units. For operating convenience, provisions have been made to allow the use of optional on-line unit record machines; e.g., an on-line IBM 7550 Card Punch may be used to produce a program deck in addition to the normal program tape output. If the 7070 or 7074 Data Processing System used for assembly has more than the minimum of four tape units, the additional tape units may be used for multiple input and/or output to reduce tape handling time between assemblies.

The Four-Tape Autocoder program may be obtained by sending a reel of magnetic tape, at least 400 feet in length, to the IBM Program Librarian. The 7070/7074 Four-Tape Autocoder will be written on the tape which will be returned. It is suggested that a duplicate of that tape be made (by a system run) as soon as it is received so that one copy of the program can be kept in reserve. The extra copy should be used only in case the working copy becomes unusable, e.g., the tape has been damaged by dropping the reel containing the working copy of the tape. The reel of magnetic tape for 7070/7074 Four-Tape Autocoder should be sent to:

IBM 7070/7074 Program Librarian
International Business Machines Corporation
590 Madison Avenue
New York 22, New York

Part I: 7070/7074 Four-Tape Autocoder Language

7070/7074 Four-Tape Autocoder Statements

Coding Sheet

All Four-Tape Autocoder statements are written on the 7070 Autocoder Coding Sheet, form X28-6417-2 (see figure 1). The coding sheet indicates by column numbers the input card format for both the Basic Autocoder and full Autocoder systems. Four-Tape Autocoder uses the same input card format as full Autocoder. An explanation of the purpose of each heading on the sheet is given below.

Heading Line

Space is provided at the top of each page to identify and date the program. The information entered in the spaces labeled "Program," "Programmed By," and "Date" is not part of the source program language and is not punched.

Page Number (Columns 1-2)

A two-character page number sequences the coding sheets. Any alphameric characters may be used, providing they are acceptable to the input/output equipment of the 7070 or 7074 system used to process the Autocoder source program. (This applies in general to the usage of alphameric characters in all Autocoder statements.) The standard collating sequence should be followed in sequencing the pages. The collating sequence and alphameric characters which are not acceptable to various input/output equipment may be found on page 9 of the IBM Reference Manual "7070 Data Processing System," form A22-7003-2.

Line (Columns 3-5)

A three-character line number sequences coding entries on the sheet. The first twenty-five lines on each sheet are prenumbered 01 through 25. Any alphameric character may be used in the units position. The collating sequence should be followed in using the units position. Also provided are five non-numbered lines at the bottom of the sheet. These may be used to continue the line numbering or for inserts.

The sequence of the cards entered into the processor will be checked by the page and line numbers punched in the source program deck. Any variation from the collating sequence will be noted on the console typewriter during the running of the assembly program.

Label (Columns 6-15)

The label column is used to represent the location of data or instructions in the machine. It may be blank or it may contain an actual or symbolic address. Only those data or instructions which will be referred to elsewhere in the program need have a label.

Operation (Columns 16-20)

The operation column contains the mnemonic representation of the operation to be performed. Actual machine operation codes are never used. The Four-Tape Autocoder mnemonic operation codes are composed of from one to five alphameric characters, and are written left-justified in the operation column. Operation codes are categorized as "imperative," "declarative" and "control" codes. A description of these three types of codes is contained on pages 8, 9 and 23.

Operand (Columns 21-75)

The operand contains the actual or symbolic address of the information which is to be acted on by a particular command. In some cases, field definition, address adjustment and indexing may be used in conjunction with the address. The operand may contain the actual data to be operated on by an instruction, referred to as a "literal." It may also be used to specify index words, electronic switches, channels, channel and unit, arm and file, inquiry and unit record synchronizers, latch numbers and alteration switches.

Identification (Columns 76-80)

Program identification is punched into columns 76-80 of all cards in the source program deck. All five positions of the identification entry must be numerical.

IBM®

FORM X28-6417-2
PRINTED IN U.S.A.

Program _____ 7070 AUTOCODER CODING SHEET Identification ____ (76 80)

Programmed by _____ Poge No. [ ] (1 2) of _____

Dote _____

Line (3 5)	Label (6 15)	Operation (16 20)	OPERAND (21 25 30 35 40 45 50 55 60 Bosic Autocoder →)	(65 70 75 Autocoder →)
0 1				
0 2				
0 3				
0 4				
0 5				
0 6				
0 7				
0 8				
0 9				
1 0				
1 1				
1 2				
1 3				
1 4				
1 5				
1 6				
1 7				
1 8				
1 9				
2 0				
2 1				
2 2				
2 3				
2 4				
2 5				

Figure 1.

Address Types

The following types of addresses may appear in the label and/or operand fields of Four-Tape Autocoder statements: blank, actual, symbolic and literal. A description of these address types and the rules governing their usage follows.

Blank

The label column may be blank if the corresponding entry is not referred to elsewhere in the program. A blank operand is valid for certain operations only. If a Priority Release command has a blank operand, the processor will insert 0097. In all other cases where a blank operand is valid, the processor will insert 0000. If a blank operand is invalid, an error message will be typed out.

Actual

An actual address is valid in the label column and in the operand. It may be from one to four digits, written left-justified on the coding sheet. An actual address in the label column will cause the corresponding instruction to be assigned that actual location. The actual location will not be reserved, however, and the contents of the processor's location assignment counter will not be changed. Hence, the programmer should be extremely cautious when using an actual label. Note that an actual label may not be used with declarative operation codes.

Line	Label	Operation	OPERAND
0 1	4 5 6	S 1	3 2
0 2	0 9 7 2	B	2 5 5 4
0 3			

Symbolic

A symbolic address is valid in the label column and in the operand. It may contain from one to ten digits or letters (no special characters), with the following restrictions: the first, or leftmost, position must be a letter; blanks may not appear within a symbol.

The asterisk, *, is a special-purpose type of symbol which is valid in the operand only. Unless an actual address has been written in the label column, the processor will assign to * the contents of the location assignment counter, which will be the location of the instruction being processed. For example, if the instruction ZA1 * has been entered with an * operand, the processor will assign to * the actual label i.e., the assembled machine instruction will be +1300094440. If an actual label has been entered with an * operand, the processor will assign to * the actual label address.

Use of the asterisk address will reduce the number of symbols required in the label column. Unless there is a note to the contrary, the special symbol * may be used as an operand address wherever this manual indicates that a symbolic address is valid.

EXAMPLES

Line	Label	Operation	OPERAND
0 1	X P L U S 2 4	Z A 1	K 2 4 A B C
0 2	W A I T	B	*
0 3	T E S T S U M	B M 3	S E C 1 4
0 4			

Literal

A literal is the actual data to be operated on by an instruction. It is valid in the operand only. A literal may be numerical or alphameric. A numerical literal is from one to ten digits preceded by a plus or minus sign. An alphameric literal is from one to five alphameric characters preceded and followed by the @ symbol.

An alphameric literal may include blanks; it may not include the @ symbol.

A special-purpose numerical literal may be used to provide address constants. This special-purpose literal, called an "adcon," is of the form ± SYMBOL. The plus or minus sign indicates a numerical literal. When the sign is followed by a symbol, the processor will find the address which has been assigned to the symbol, and then treat the sign and the address as a four-digit numerical literal. Note that the special symbol * may not be used as an adcon.

The processor will assign a storage address and field definers to a literal, and build that address and field definition into the instruction being processed. Literals will be packed into words to conserve storage. They will be positioned in the object program and symbolic-language program as specified by Litorigin Control entries in the source program (see page 25).

EXAMPLE

Line	Label	Operation	OPERAND
0 1			+1,2,3,0,5,6,7
0 2			-2
0 3			+1,8
0 4			@,A,B,C,D,E,@
0 5			@,A,B, @
0 6			@,A,@
0 7			+M,A,N,N,O
0 8			

Certain imperative instructions (e.g., Priority Control, Index Word Load, Tape Write) operate on full words and do not permit field control. If a literal which is less than ten digits, or an adcon, is used as the operand of any of these instructions, it will be converted to ten-digit form by being right-justified in a word. Thus, if –50 is entered as the address portion of a Tape Write instruction (TW), the area to be written from will be defined by the record definition word –0000000050. Also, assume that ATABLE is the label of an instruction or data occupying location 2000. If the adcon –ATABLE is entered as the address portion of an Index Word Load and Interchange command (XLIN), the contents of the specified index word will be replaced by -0020000000.

Field Definition

Field definition, if required, is written immediately after the operand address. The field definers, digits 0-9, are enclosed in parentheses and the starting position of the field is separated from the ending position by a comma. When operating on a single digit, the comma and ending digit position may be omitted. Field definers may be omitted entirely when operating on a whole word or a field defined by a declarative operation (see pages 9 through 22).

The listing of Autocoder imperative operation codes in Appendix B indicates those operation codes which permit field definition to be associated with the address. Field definition may be used with actual and symbolic addresses, and with literals, if applicable.

EXAMPLE

Line	Label	Operation	OPERAND
01			*(8,9)
02			MANNO(0,2)
03			1024(,2,,8)
04			ALPHA(5)
05			MANNO
06			1024
07			+156827(3,,4)
08			-SYMBOL(2,,3)
09			

When used with literals, field definition will be relative to the literal itself. Thus, the entry +156827(3, 4) refers to the fourth and fifth digits of the literal, i.e., +82. Also, the entry -SYMBOL(2, 3) refers to the third and fourth digits of the location assigned to SYMBOL; i.e., if location 2332 has been assigned to SYMBOL, -SYMBOL(2, 3) would be equivalent to -32. (See "Relative Field Definition" on page 14.)

Address Adjustment

Address adjustment may be used to allow the programmer to refer to an entry which is a given number of locations away from a symbolic address. Its usage will thus reduce the number of symbols necessary for the source program. Address adjustment is indicated by writing immediately after the symbolic address, or field definers, if any, a plus or minus sign followed by from one to four digits.

Address adjustment is permitted with all symbolic addresses, excepting the single-address operand of a DRDW statement (see page 20). It is not permitted with actual addresses or literals. The programmer should be careful when using address adjustment. For example, inserts and deletions of program entries could change addresses in such a way that *+10 should now be *+9.

EXAMPLES

Line	Label	Operation	OPERAND
01			MANNO+15
02			MANNO-2
03			MANNO(0,2)+15
04			*-11
05			

If location 2150 has been assigned to the symbolic address MANNO, then 2165 will be assigned to MANNO+15, and 2148 will be assigned to MANNO-2. The entry MANNO(0, 2)+15 refers to the first three digit positions of location 2165. Similarly, if 2196 is the location of an instruction containing *-11 as an operand, then location 2185 will be assigned to this operand.

Index Words

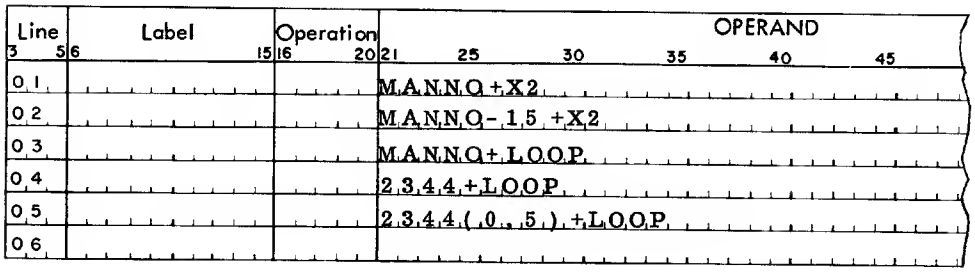
Indexing

An indexing word may be written symbolically or in the form xn; it is always preceded by a plus sign. x denotes an index word and n is the number (1-99) of the index word. The x is required so that Autocoder can distinguish between ad-

dress adjustment and indexing. (When used for other than indexing purposes, the form xn will be considered a symbol.) The indexing word always follows the operand address, after field definers and address adjustment, if any.

Symbolic and actual addresses of all imperative instructions are indexable. Literals are not indexable. With the exception of Branch Control and End Control operations (see pages 26 and 27), indexing is not permitted in declarative and control operations.

EXAMPLES



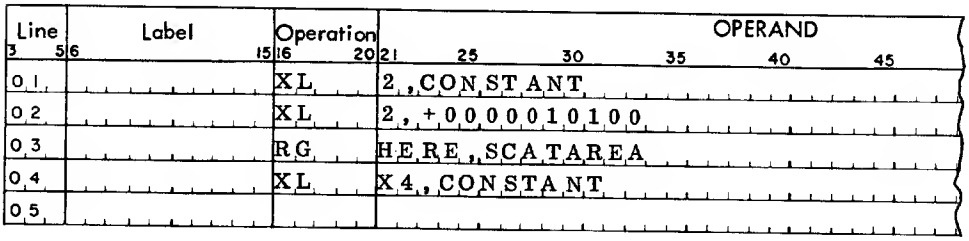
Line	Label	Operation	OPERAND
01			MANNO+X2
02			MANNO-15+X2
03			MANNO+LOOP
04			2,3,4,4,+LOOP
05			2,3,4,4,(,0,,5,),+LOOP
06			

Autocoder will interpret +x2 as index word 2, and +LOOP as the symbolic designation of an index word.

Uses Other Than Indexing

When an index word is to be specified for other than indexing purposes, i.e., in index word commands such as Index Word Load (XL) and Index Word Load and Interchange (XLIN), or in commands such as Record Gather (RG) and Record Scatter (RS), the index word may be written in actual or symbolic form and appears first in the operand. In these cases, xn would be interpreted as the symbolic designation of an index word.

EXAMPLES



Line	Label	Operation	OPERAND
01		XL	2,,CONSTANT
02		XL	2,,+0000010100
03		RG	HERE,,SCATAREA
04		XL	X4,,CONSTANT
05			

The first two examples refer to an actual index word, i.e., index word 2. The last two examples refer to symbolic index words. Note that Autocoder may assign any index word as x4 in the last example because x4 is a symbolic index word; x4 does not mean index word 4 when used in this manner.

Index Word Reservation

The processor reserves all actual index words used in the source program in Phase 2. The unreserved index words (excepting index words 97, 98 and 99) will be assigned in numerical sequence to symbolic index words during Phase 3. Index words will also be reserved during Phase 2 if the location assigned to an imperative operation or if any location defined by a declarative operation falls within the range 0001 to 0096. If the programmer desires, actual index words may be assigned to symbols by use of the declarative operation EQU (see page 21); index words thus assigned will be reserved in Phase 2.

It is possible that the programmer may wish to reserve a particular group of index words for subsequent use in his program. This may be accomplished by inserting an Origin Control operation code entry (see page 23) whose operand is an actual index word address. Succeeding entries will be assigned sequential

locations starting at the specified index word address. Unlabeled succeeding entries will reserve the index words they occupy; labeled entries will both reserve and name the index words.

Electronic Switches

Electronic switches may be coded in actual or symbolic form. Switches referred to by their actual number (1-30) will be automatically reserved in Phase 2. The unreserved switches will be assigned in numerical sequence to symbolic switches during Phase 3. Actual switches may be assigned to symbols by use of the declarative operation EQU (see page 22); switches thus assigned will be reserved in Phase 2. Unlike index words, electronic switches will *not* be reserved if the location assigned to an imperative operation or if any location defined by a declarative operation falls within the range 0101-0103.

EXAMPLES

Line	Label	Operation	OPERAND
0 1		BES	19,COMPUTE
0 2		BSF	END,LOOP
0 3		ESN	28
0 4			

Input/Output Unit and Alteration Switch Designations

The following items may be specified in actual or symbolic form in the operands of those instructions which refer to the particular items: channel, combined channel and unit, combined arm and file, unit record synchronizers, inquiry synchronizers and alteration switches. The declarative operation EQU is used to equate item numbers to symbolic names (see page 23).

EXAMPLES

Line	Label	Operation	OPERAND
0 1		TR	MASTER,INPUT
0 2			
0 3		PTW	14,,-0012421260
0 4			
0 5		TSB	25,FILEBACK
0 6			

Remarks and Comments

Remarks may be included anywhere in the operand, provided they are separated by at least two blank spaces from the operand of the instruction.

The programmer may insert lines of descriptive information in the program by placing an asterisk (*) in column 6 of the label column. Comments inserted in this way will appear in the symbolic output, but will not in any way affect the operation of the program. A comments card produces no entry in the object program. Comments cards may be placed anywhere in the source program except as follows:

1. A comments card may *not* be placed between the DTF entries describing the files used in the program.

2. A comments card may *not* be placed between subsequent entries under a DTF.
3. A comments card may not be placed between continuation cards of a macro-instruction.

Any part of the label, operation or operand columns may be used for the description. Comments cards are useful as descriptive headings for various sections of a program, such as operating instructions, or where the operand column of an instruction does not allow enough room for necessary remarks.

EXAMPLES

Line	Label	Operation	OPERAND
0 1	*NET PAY CALCULATION		
0 2		ZA 1	GROSS PUT GROSS PAY IN ACC1
0 3		S 1	TAX DEDUCT INCOME TAX
0 4		S 1	FICA DEDUCT FICA
0 5			

Imperative Operations

Autocoder imperative operations are direct commands to the 7070 or 7074 to operate on data, constants or other instructions, using the available components of the machine. Each machine-language instruction has a corresponding unique imperative operation code. The codes consist of from one to five alphameric characters and are mnemonic in form. For example, the Autocoder equivalents of the Branch, Unit Record Read, Unit Record Punch, and Shift Right Split commands are B, UR, UP, and SRS, respectively.

Only those imperative operations which will be referred to elsewhere in the program need have labels. Operands may contain the symbolic or actual address of data which is to be operated on by the instruction, with or without address adjustment and indexing; they may contain the actual data itself (literals). Field definition is permitted in the operand of any imperative operation whose corresponding machine-language instruction permits it; i.e., digit positions 4 and 5 are reserved for field control or are unused. Note that address adjustment is not permitted with literals and actual addresses, either with or without field definition.

The correct order of entry for operand addresses is as follows: operand address, field definers, address adjustment, indexing. For example,

Line	Label	Operation	OPERAND
0 1		CD	FIELDA(,2),+4,+IWORD,6
0 2		.	
0 3		.	
0 4		XA	IWORD,FIELDA+4+IWORD
0 5			

Assume in the above illustrations that FIELDA has been defined as word 2000 and that the indexing portion of index word IWORD contains 0100. In the first example, digit position 2 of location 2104 will be compared to a "6." In the second example, 2104, considered plus, will be algebraically added to the number 0100 in the indexing portion of IWORD.

An alphabetic list of Autocoder imperative operation codes is contained in Appendix B. The list is preceded by an operand symbol key, which indicates what is permissible in the operand field and the order in which the information must be entered on the coding form.

Following is a section of a payroll routine which illustrates typical Autocoder coding. Note that the remarks may appear anywhere in the operand, provided two blank spaces separate the remarks from the operand of the instruction.

Line	Label	Operation	OPERAND	Basic
01	CALCTAX	ZA,3	TC DETERMINE IF	
02		M	-1,3,0,0, PAY IS	
03		A,2	GROSS, TAXABLE	
04		BM,2	NO TAX BRANCH IF NO TAX	
05		ZA,3	9,9,9,2	
06		M	+1,8, CALCULATE TAX	
07		SRR	2	
08		B	*+2	
09	NOTAX	S,2	9,9,9,2, CLEAR ACCUM 2	
10		ST,2	WITHHOLD STORE TAX AMOUNT	
11				

Declarative Operations

Autocoder declarative operations are statements to the processor which provide it with the necessary information to complete the imperative operations properly. Declarative operations are never executed in the object program and should be separated from the program instruction area, placed preferably at its beginning or end. Otherwise, special care must be taken to branch around them so that the program will not attempt to execute something in a data area as an instruction.

Four-Tape Autocoder includes the following four declarative operation codes: DA (Define Area), DC (Define Constant), DRDW (Define Record Definition Word), and EQU (Equate). DA and DC operations require more than one entry. The DA code is used to name and define the positions and lengths of fields within an area. The DC code is used to enter constants into the object program. Since the 7070 and 7074 make use of record definition words (RDWs) to read, write, and otherwise examine blocks of storage, the DA and DC codes provide the option of generating RDWs automatically. When so instructed, Autocoder will generate one or more RDWs and assign them successive locations immediately preceding the area with which they are associated. An RDW will be of the form ±00xxxxyyyy, where xxxx is the starting location of the area and yyyy is its ending location. These addresses are calculated automatically by the processor.

In some cases, it may be more advantageous to assign locations to RDWs associated with DA and DC areas in some other part of storage, i.e., not immediately preceding the DA or DC areas. The operation code DRDW may be used for this purpose. The DRDW code may also be used to generate an RDW defining any area specified by the programmer.

The declarative operation code EQU will permit the programmer to equate symbolic names to actual index words, electronic switches, arm and file numbers, channel and tape unit numbers, alteration switches, etc., and to equate a symbol to another symbol or to an actual address.

A detailed description of the formats and functions of the Four-Tape Autocoder declarative operations follows below.

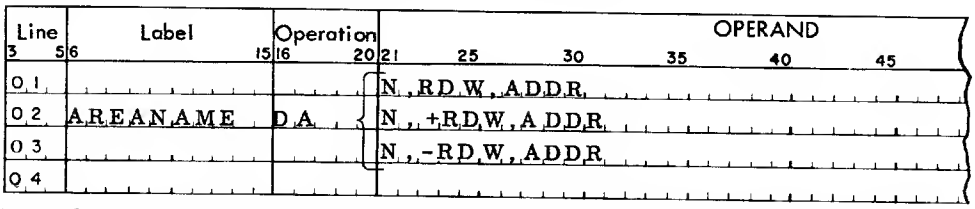
DA—Define Area

The declarative operation DA may be used to reserve and define any portion of storage, such as an input, output or work area, or an area which will contain more than one record, all of which are identical in format. The DA operation instructs the processor with regard to the positions, lengths and names of fields which make up the area being defined. The processor will then automatically assign locations and field definers, so that the fields may be conveniently referred to by name in instruction operands. Thus, the programmer need not be concerned with the actual locations of the fields within storage.

A DA operation consists of a header line and one or more succeeding entries with blank operation columns. The DA header line is used to initiate the reservation of a portion of storage. The succeeding entries define the fields within this portion and specify the amount of storage to be reserved.

DA Header Line

The general format of the DA header line is as follows:



Use of a label is optional. (An actual label may not be used.) Also, "RDW" and a specified address, ADDR, may be omitted from the operand. If RDW is omitted, separating commas must be punched in two consecutive columns between N and ADDR. If ADDR is omitted, the preceding comma need not be punched. Also, if both RDW and ADDR are omitted, both commas may be omitted.

Area Number

N specifies the number of identical storage areas to be reserved by the DA operation, only the first of which will be defined. For example, suppose that records of identical format are to be read into storage in blocks of ten. The programmer would then enter a DA header line, with N equal to 10, followed by succeeding entries which specify the starting and ending positions and names of the fields in one record.

N may be from 1 to 999. The number of storage words to be reserved for the entire area will be N times the number of words reserved by succeeding entries, i.e., N times the number of words reserved for one record area. The maximum number of words which may be reserved for one record is 999. Frequently, N will equal 1, as in a work area.

Record Definition Words

Record definition words are required by the 7070 and 7074 for reading in and writing out data and for moving blocks of data within storage. If "RDW" is written in the operand of the DA header line, the processor will automatically generate N RDWs associated with the N defined areas. These RDWs will be assigned N locations immediately preceding the first word of the area. The label of the DA header line will be made equivalent to the first RDW generated by the processor; i.e., if the label appears as the operand address of an imperative instruction, it will refer to the location of the first RDW. The first word of the area (and any field which occupies this first word) may be referred to by the DA label +N.

If RDW is not preceded by a sign, all generated RDWs will be plus except the last, which will be minus. If a + or – sign precedes RDW, all generated RDWs will be given the indicated sign.

If RDW is not written in the operand, RDWs will not be generated and the label of the DA header line will be made equivalent to the first word of the area.

For example, suppose that an area is to contain a group of four 10-word records and that the following DA header line is entered, the fields within a record being defined by succeeding entries:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
0 1	AJAX	DA	4,,RDW
0 2			
0 3			
0 4			

Assume that Autocoder's location assignment counter contains 1000 when the above DA operation is encountered. The processor will generate the following RDWs associated with the four record areas and store them in locations immediately preceding the first record area:

Location	RDW
1000	+0010041013
1001	+0010141023
1002	+0010241033
1003	−0010341043

The label AJAX will refer to the first RDW; e.g., the imperative instruction ZA1 AJAX will be translated in the object program as +1300091000. The first word of the area, occupying location 1004, may be referred to by AJAX+4. If RDW had not been entered in the DA header line, the first word of the area would be stored in location 1000 and would be referred to by the label AJAX.

Relative Addressing

The fields defined in succeeding entries may be assigned addresses relative to an address (ADDR) specified in the operand of the DA header line. For example, assume that a relative address of 0 is specified. Then any field which occupies the first, second or third word, etc., of the area will be assigned relative addresses of 0000, 0001, 0002, etc., respectively. Thus, if FIELDC is the name of a field occupying the third word of the area, the instruction ZA1 FIELDC(0, 4)+x11 would be translated in the object program as +1311040002.

The relative address may be written in actual or symbolic form. If the latter, the symbol must have appeared as a label earlier in the program sequence. (Note that the special symbol * may not be used.) Address adjustment is permitted with a symbolic address.

The chief use of relative addressing would be to facilitate indexing if an area contains grouped records and it is desired to operate on fields within successive records. This may be accomplished by making the fields relative to 0 and indexing the instructions which refer to the fields by an index word containing, successively, the RDWs associated with the records. For example, assume that an area is to contain a group of four 10-word records (as in the previous example). Suppose that it is desired to perform the following operation on three fields named FIELDA, FIELDB and FIELDC, which occupy the first, second and third words, respectively, of each record: subtract the contents of FIELDB from the contents of FIELDA and store the result in FIELDC. This could be accomplished by the following coding:

Line 3 5	Label 6 15	Operation 16 20	21 25 30 35
0 1	INAREA	DA	4,,+RDW,,0
0 2	(Succeeding entries define		
0 3	the fields within a record)		
0 4		•	
0 5		•	
0 6		XL	3,,+0,0,0,0,0,0,0,0,0,3,
0 7	LDRDW	XL	2,,INAREA+X3
0 8		ZA 1	FIELDA+X2
0 9		S 1	FIELDB+X2
1 0		ST 1	FIELDC+X2
1 1		BI X	3,,LDRDW
1 2			

Location	Assembled Instruction
2000	+4500034432
2001	+4503021000
2002	+1302090000
2003	-1402090001
2004	+1202090002
2005	+4900032001

Assume that Autocoder's assignment counter contains 1000 when the DA operation is encountered, 2000 when the first instruction of the routine is encountered, and the literal +0000000003 has been assigned location 4432. The corresponding assembled machine-language instructions would be as shown above, to the right of the coding entries.

Succeeding Entries

Lines directly below the DA header line contain the names and starting and ending positions of fields within an area (or first record of an area if N is greater than 1). These lines have blank operation columns.

When defining the fields within an area, it is important to remember how the 7070 and 7074 operate. The 7070 and 7074 have the flexibility of operating on a field within a word. Thus, when the control part of a machine-language instruction contains field definers, only that part of the word specified by the field definers is used in the operation.

The position of a field within an area defined by a DA operation is indicated in the operand column by its starting digit position in the area, a comma, and its ending digit position in the area. If a field occupies one digit position, the comma and ending digit position may be omitted. The first word of the area is referred to as word 0, the second as word 1, the third as word 2, etc. Digits within a word are numbered 0 through 9, from left to right. A digit position of an area is defined as a word number (one to three digits) followed immediately by a digit number (one digit). For example, digit 6 of the ninth word of the area (word number 8) corresponds to digit position 86 of the area. If the area reserved by a DA operation is considered to be a consecutive string of digits, the first being 00, the second 01, etc., the digit position of any digit will be its placement in the entire string of digits composing the area.

The order of entries of fields under a DA is irrelevant: i.e., fields need not be in ascending storage position order.

The following example illustrates the procedure for entering the positions of fields under a DA header line:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
0 1	GRUMP	DA	1
0 2	NAME 1		0 0 , , 1 6
0 3	NAME 2		1 7 , , 1 9
0 4	NAME 3		2 0 , , 2 9
0 5	NAME 4		5 3 , , 5 7
0 6	NAME 5		4 8
0 7			7 9
0 8			

The above entries define an eight-word area, beginning with three fields of lengths 17, 3 and 10 digits, respectively. A gap of 18 digits then occurs, followed by the one-digit NAME5. A second gap of 4 digits occurs, and then the five-digit field NAME4. (Note that NAME4 and NAME5 are not in ascending storage position order.) The rest of the area is unnamed. The one-digit field with a blank label in digit position 79 establishes that the area occupies eight words. Note that a field which is unreferenced by the program and which terminates the area whose fields are being defined may be entered with a blank label, as in this example. Otherwise, only named fields need be entered under a DA.

Subfields are indicated to any desired degree of depth by simply entering their starting and ending digit positions, which may overlap, fall within or bridge other fields. Again, the order of entry is irrelevant. To illustrate, the following entries might appear under a DA header line:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
0 1	DECADE		7 8
0 2	DAY		7 6 , , 7 7
0 3	YEAR		7 8 , , 7 9
0 4	MONTH		7 4 , , 7 5
0 5	DATE		7 4 , , 7 9
0 6			

A field which bridges two or more consecutive words cannot be operated on with field definers in one instruction. A bridging field may be given one name; however, if the name appears as the operand address of an instruction, only that part of the field which occupied the first word will be operated upon.

The following rule applies to fields which cross word boundaries: If a bridging field starts with digit 0 of a word, any part (which does not cross word boundaries) may be referred to by the proper use of address adjustment and field definition. If a bridging field starts with a digit other than a 0 and it is desired to refer to parts of the field that lie beyond the first word boundary, names must be given to these parts on separate entries below the DA header line. To illustrate this rule, the following examples present both methods of referring to data in a field that crosses word boundaries. In both examples, the field that crosses word boundaries is labeled FULLNAME and it is assumed that the first sixteen digits contain the first name, the second sixteen digits the middle name and the remaining thirty digits the last name.

EXAMPLE 1

The following DA entry includes a field that crosses word boundaries and begin in digit position 0 of a word:

Line	Label	Operation	OPERAND
01	HISTORYREC	DA	1,RDW
02	DEPTNO		00,05
03	EMPLOYEENO		10,19
04	FULLNAME		20,81
05	RESTOFREC		82,1820
06			

Any part of the FULLNAME field which does not cross word boundaries may be referred to using field definition and address arithmetic because FULLNAME begins in digit position 0 of a word. The initials of the first, middle and last names in the FULLNAME field can be referred to as FULLNAME (0, 1), FULLNAME (6, 7) + 1 and FULLNAME (2, 3) + 3, respectively.

EXAMPLE 2

The following DA entry includes a field that crosses word boundaries and begins in digit position 6 of a word:

Line	Label	Operation	OPERAND
01	HISTORYREC	DA	1,RDW
02	DEPTNO		00,05
03	EMPLOYEENO		06,15
04	FULLNAME		16,77
05	RESTOFREC		78,1780
06	MIDDLENAME		32,47
07	LASTNAME		48,77
08			

Labels have been assigned to the parts of the FULLNAME field containing the middle name and last name; those parts cannot be referred to using address arithmetic because FULLNAME does not begin in digit position 0. The initials of the first, middle and last names can be referred to as FULLNAME (0, 1), MIDDLENAME (0, 1) and LASTNAME (0, 1), respectively.

Relative Field Definition

The primary function of the DA declarative operation is to instruct the processor as to the positions and lengths of fields within an area, thereby allowing the processor to assign storage locations and field definers automatically when a field is referred to by name in the operand of an instruction. Sometimes it may be desirable to refer to a portion of a field which has been defined by a declarative operation. For the convenience of the programmer, this is done relative to the field itself, so that it is not necessary to remember the actual digit positions of the field.

To illustrate, assume that the name CUSTNO is a field of seven digits, defined under a DA as follows:

Line	Label	Operation	OPERAND
01	INPUTAREA	DA	1
02	CUSTNO		03,09
03			

A field might occupy only the low-order digit positions of a word, as does CUSTNO above. However, positions within the field itself are numbered starting with 0. If the programmer wishes to refer to the three high-order positions of CUSTNO, he would refer to positions 0, 1 and 2 by placing the entry CUSTNO(0, 2) in the operand. During assembly, the processor will convert the field definers relative to CUSTNO to field definers which refer to actual digit positions of a word. Thus, during assembly, the processor will convert CUSTNO(0, 2) to the actual digit positions 3, 4 and 5 of the word of which CUSTNO is a portion. If the storage location of this word is 1001, symbolic instructions using CUSTNO and their assembled equivalents might be as follows:

Symbolic Instruction	Assembled Instruction
ZA1 CUSTNO(0, 2)	+1300351001
ZA1 CUSTNO(4, 5)	+1300781001
ZA1 CUSTNO	+1300391001

CUSTNO might instead be defined as a full word, as follows:

Line	Label	Operation	OPERAND
01	INPUTAREA	DA	1
02	CUSTNO		00,09
03			

In this case, any field definition following CUSTNO would be both relative and actual. If the storage location of this word is 1001, symbolic instructions using CUSTNO and their assembled equivalents might be as follows:

Symbolic Instruction	Assembled Instruction
ZA1 CUSTNO(0, 2)	+1300021001
ZA1 CUSTNO(4, 5)	+1300451001
ZA1 CUSTNO	+1300091001

In the example on page 13 illustrating how subfields are entered, the subfield DAY would be equivalent to DATE(2, 3).

DC—Define Constant

The declarative operation DC may be used to enter numerical, alphameric and address constants (adcons) into the object program, and to assign names to constants for ease of reference. Like the DA operation, the DC operation consists of a header line and one or more succeeding entries with blank operation columns. The DC header line directs the processor to assign an area in storage to the constants entered directly below.

DC Header Line

The format of the DC header line is as follows:

Line	Label	Operation	OPERAND
01	ANYNAME	DC	+RDW (or blank)
02			

Use of a label is optional. (An actual label may not be used.) If plus or minus "RDW" is written in the operand, the processor will generate an RDW with the indicated sign, which defines the area in storage containing the constants written under the DC header line. The RDW will be stored in the location immediately preceding the "constant" area. The label of the DC header line will refer to the RDW.

If the operand is blank, no RDW will be generated and the DC label will refer to the first word of the "constant" area.

Constants are entered in the operands of succeeding entries, as described below. Note that any combination of numerical and alphameric constants may be entered under a DC, provided the rules for each are obeyed.

Numerical Constants and Adcons

Like numerical literals, numerical constants may be from one through ten digits preceded by a plus or minus sign. As previously mentioned, adcons are of the form ±SYMBOL.

Numerical constants and adcons may or may not be packed into words, as the programmer desires. Constants written without field definition will be right-justified in separate words. Constants written with field definers will be packed into words according to the following rules:

1. A change in sign will start a new word.
2. Field definition which would cause overlapping in the same word will force the overlapping constant into a new word.

The programmer may enter more than one numerical constant or more than one adcon on a line, but not a mixture of both. The constants or adcons will be right-justified in separate words. Field definition is not permitted when more than one constant or adcon is entered on a line.

Constants may be assigned separate names for ease of reference in the source program. The named constants may or may not be packed into words, as the programmer desires.

Following are examples which illustrate the various ways in which numerical constants and adcons may be entered under a DC header line. Assume in all these examples that Autocoder's location assignment counter has reached 1000 when the DC operation is encountered.

EXAMPLE 1

The following entry

Line	Label	Operation	OPERAND
0 1	CONAREA	DC	
0 2			+152 +38 +97 −21 −304 +16
0 3			

will cause the indicated constants to be right-justified in separate words. Autocoder will make the following assignments:

Symbol	Field Definers	Address	Contents
CONAREA	0,9	1000	+0000000152
		1001	+0000000038
		1002	+0000000097
		1003	−0000000021
		1004	−0000000304
		1005	+0000000016

In this example, the operand of the DC header line is blank. Therefore, the label CONAREA will refer to the first constant, +152; CONAREA+1 will refer to +38; CONAREA+2 will refer to +97, etc.

EXAMPLE 2

If the programmer desired a +RDW to be associated with the "constant" area of the previous example, he could write

Line	Label	Operation	OPERAND
01	CONAREA	DC	+RDW
02			+152,+38,+97,-21,-304,+16,
03			

Autocoder would then make the following assignments:

Symbol	Field Definers	Address	Contents
CONAREA	0,9	1000	+0010011006
		1001	+0000000152
		1002	+0000000038
		1003	+0000000097
		1004	−0000000021
		1005	−0000000304
		1006	+0000000016

The label CONAREA will refer to the RDW defining the "constant" area; CONAREA+1 will refer to the first constant, +152; CONAREA+2 will refer to + 38; etc.

EXAMPLE 3

The following coding shows how separate names may be assigned to constants:

Line	Label	Operation	OPERAND
01	CONAREA	DC	
02	C1,		+152,
03	C2,		+38,
04	C3,		+97,
05	C4,		-21,
06	C5,		-304,
07	C6,		+16,
08	C7,		-CONAREA,
09	C8,		-C5,
10			

In this case, Autocoder will make the following assignments:

Symbol	Field Definers	Address	Contents
CONAREA C1	0,9	1000	+0000000152
C2	0,9	1001	+0000000038
C3	0,9	1002	+0000000097
C4	0,9	1003	−0000000021
C5	0,9	1004	−0000000304
C6	0,9	1005	+0000000016
C7	0,9	1006	−0000001000
C8	0,9	1007	−0000001004

EXAMPLE 4

The following coding shows how constants may be assigned separate names and packed into words:

Line	Label	Operation	OPERAND
0 1	CONAREA	DC	
0 2	C1		+1523(0,2)
0 3	C2		+38(3,4)
0 4	C3		+97(5,6)
0 5	C4		-21(0,1)
0 6	C5		-3040(6,8)
0 7	C6		+16
0 8	C7		-CONAREA(2,5)
0 9	C8		-C5(6,9)
1 0			

Autocoder will make the following assignments:

Symbol	Field Definers	Address	Contents
CONAREA	0,9	1000	+1523897000
C1	0,2		
C2	3,4		
C3	5,6		
C4	0,1	1001	−2100003040
C5	6,8		
C6	0,9	1002	+0000000016
C7	2,5	1003	−0010001001
C8	6,9		

Alphameric Constants

An alphameric constant is preceded and terminated by the @ sign. The @ sign may be used as a character within an alphameric constant *only* if it immediately precedes the terminal @ sign. Alphameric constants are always packed into words. Therefore the programmer (in writing a message, for example) must include all blanks in the proper places.

Autocoder will extract groups of five characters from the alphameric constant entered under a DC and store them in separate words. The characters will be converted to double-digit form and stored successively beginning in the high-order position of a word, i.e., digit 0. If the total number of characters in the constant is not a multiple of five, blanks (double-digit zeros) will be stored in the low-order positions of the last word used. The sign of all the words used to contain the characters will be alpha.

EXAMPLE 1

Assume in this and subsequent examples that Autocoder's location assignment counter has reached 1000 when the DC operation is encountered.

Line	Label	Operation	OPERAND	Basic Autocoder
0 1	EOFMSG	DC	-RDW	
0 2			@REMOVE TAPE ON UNIT 00 AND MOUNT NEXT@	
0 3				

Autocoder will make the following assignments:

Symbol	Field Definers	Address	Contents
EOFMSG	0,9	1000	–0010011008
		1001	R E M O V
		1002	E T A P
		1003	E O N
		1004	U N I T
		1005	0 0 A N
		1006	D M O U
		1007	N T N E
		1008	X T

Note that the label EOFMSG refers to the RDW defining the constant area.

EXAMPLE 2

The constant in the previous example may be written as follows if it is desired to name a portion:

Line	Label	Operation	OPERAND
0 1	EOFMSG	DC	-RDW
0 2			@REMOVE TAPE ON UNIT @
0 3	TAPENO		@0 0@
0 4			@ AND MOUNT NEXT@
0 5			

Autocoder will make the following assignments:

Symbol	Field Definers	Address	Contents
EOFMSG	0,9	1000	–0010011008
		1001	R E M O V
		1002	E T A P
		1003	E O N
		1004	U N I T
TAPENO	0,3	1005	0 0 A N
		1006	D M O U
		1007	N T N E
		1008	X T

It will be the programmer's responsibility to insure that any named portion does not overlap, or bridge, words.

The record mark sign (≠) may not be entered on the coding sheet. If desired, a record mark (80 in double-digit code) may be generated in an alphameric constant by entering the letter R immediately following the terminal @ sign. If the number of characters in the constant is not a multiple of five and the terminal @ is followed by the character R, a record mark will be placed in digits 8-9 of the last word used. If the number of characters is a multiple of five, the record mark will be placed in digits 8-9 of the next word preceded by eight zeros (four alphabetic blanks). For example, if the following coding is entered,

Line	Label	Operation	OPERAND
0 1		DC	
0 2	C 1		@A B C @R
0 3	C 2		@A B C D E @R
0 4			

Autocoder will make the following assignments:

Symbol	Field Definers	Address	Contents
C1	0,9	1000	ABC ≠
C2	0,9	1001	ABCDE
		1002	≠

The record mark in location 1000 could be referred to by C1 (8, 9); the record mark in location 1002 could be referred to by C2 (8, 9) + 1. The following coding shows how a record mark may be entered and named so that field definers are not necessary when operating on it:

Line	Label	Operation	OPERAND
0 1		DC	
0 2	BLANKS		@ @
0 3	RECMARK		@@R
0 4			

Autocoder will make the following assignments:

Symbol	Field Definers	Address	Contents
BLANKS	0,7	1000	≠
RECMARK	8,9		

The record mark may be referred to directly by the label RECMARK.

DRDW—Define Record Definition Word

The declarative operation DRDW may be used to generate an RDW defining an area of storage specified by the programmer. It may also be used to generate an RDW(s) associated with an area defined by a DA or DC operation and to place it in any desired portion of storage. The formats for these two functions of the DRDW operation are presented below. Note that an actual label may not be used with a DRDW operation.

A DRDW entry with the format

Line	Label	Operation	OPERAND
0 1	ANYSYMBOL	DRDW	±ADDR1,ADDR2
0 2			

will cause the processor to generate an RDW defining the area from ADDR1 through ADDR2. The sign of the RDW will be the sign entered in the operand. ADDR1 and ADDR2 may be two actual or two symbolic addresses, or an actual and a symbolic address. The RDW may be referred to by the label of the DRDW entry.

As previously mentioned, writing "RDW" on a DA or DC header line will cause the processor to generate an RDW(s) associated with the DA or DC entry and to assign it a storage location immediately preceding the defined area. Sometimes it may be advantageous to generate an RDW(s) associated with a DA or DC operation and to place it in another portion of storage. This may be accomplished by entering

Line	Label	Operation	OPERAND
0 1			+SYMBOL
0 2	ANYSYMBOL	DRDW	-SYMBOL
0 3			SYMBOL
0 4			

where SYMBOL must be the label of some DA or DC header line which has appeared earlier in the program sequence. If SYMBOL is the label of a DA header line, N RDWs will be generated, where N is the N entered on the DA header line. If SYMBOL is the label of a DC header line, one RDW will be generated; the operand of a DC header line has an implied N equal to 1.

The operand +SYMBOL will produce N +RDWs; -SYMBOL will produce N -RDWs; and SYMBOL will produce N RDWs, all plus except the last which will be minus. The label ANYSYMBOL of the DRDW will refer to the first RDW; ANYSYMBOL+1 will refer to the second RDW, etc. Address adjustment is not permitted with this type of DRDW entry.

EQU—Equate

The declarative operation EQU may be used to equate a symbol to:

1. An actual or symbolic address.
2. A channel, combined channel and tape unit, combined arm and file, unit record synchronizer, inquiry synchronizer or alteration switch number.
3. An index word or electronic switch number.

The EQU operation allows the programmer to use easily remembered names instead of actual machine numbers in his program. Also, it is possible that two different symbols may be assigned to the same item in different parts of a program. The EQU operation may be used to equate the second symbol to the first. Note that an EQU entry may be inserted anywhere in the source program (in contrast to the other declarative operations, which must be separated from the program instruction area).

The method of coding each of the three above uses of the EQU operation is described below.

Actual or Symbolic Address

The symbol to be equated is written in the label column. The operand may contain an actual or symbolic address, with or without field definers. The special symbol * is not permitted. If a symbol is written in the operand, it must have appeared as a label earlier in the program sequence.

EXAMPLE

Line	Label	Operation	OPERAND
01	CLASS	EQU	CUSTNO(0,1)
02			

The above entry will cause the processor to assign to the symbol CLASS the same location previously assigned to CUSTNO; the two high-order positions of CUSTNO will be assigned as the field definition (see "Relative Field Definition" on page 14).

Note that the EQU operation does not allow the transitive relation

Line	Label	Operation	OPERAND
01	A	EQU	B
02		.	
03		.	
04	B	EQU	C
05			

Statements of this form are not acceptable because they will not result in a correct location assignment.

Two other methods are suggested for assigning two different symbolic names to the same field. One is to list both names under the same DA, repeating the starting and ending digit positions of the field. The other is to place the second name under a separate DA that is made equivalent to the first by an Origin Control entry. An example of the first method is as follows:

Line	Label	Operation	OPERAND
01		DA	1
02	FIELDA		19,,27
03	FIELDB		44,,83
04			
05			
06	A		19,,27
07	B		44,,83
08			

In this example, A is made equivalent to FIELDA and B is made equivalent to FIELDB.

An example of the second method is as follows:

Line	Label	Operation	OPERAND
01	DANAME	DA	1
02	FIELDA		2,,9
03			
04			
05	ORIGIN	CNTRL	DANAME
06		DA	1
07	FIELDB		2,,9
08			

In this example FIELDA is made equivalent to FIELDB by assigning both fields to the same location in storage.

Index Word or Electronic Switch Number

The label column contains the name to be assigned. The operand contains the one- or two-digit index word (1-99) or switch number (1-30), followed by a comma and the letter x or s, respectively. An index word may be field defined. The index word or switch that is equated to the label will be reserved in Phase 2; i.e., it will be passed over on Autocoder's automatic assignment of index words and electronic switches in Phase 3.

EXAMPLES

Line	Label	Operation	OPERAND
01	LOOPCOUNT	EQU	1,X
02			
03			
04	IXWORD	EQU	52(2,,5),X
05			
06			
07	SWITCHA	EQU	25,S
08			

The first entry will assign the name LOOPCOUNT to index word 1. The second entry will assign the name IXWORD to the indexing portion of index word 52. The third entry will assign the name SWITCHA to electronic switch 25.

Channel, Channel and Unit, Arm and File, Unit Record and Inquiry Synchronizers, Alteration Switches

The label column contains the name to be assigned and the operand contains the number of the item, followed by a comma and an explanatory code character. The explanatory code characters used in EQU statement operands are as follows:

Item	Code	Item	Code
Tape Channel and Unit	CU	Unit Record Synchronizer	
Tape Channel	C	Reader	R
Disk Storage Arm and Unit	AF	Printer	W
Index Word	X	Punch	P
Electronic Switch	S	Inquiry Synchronizer	Q
Alteration Switch	SN	Typewriter	T
Unit Record Latch	I		

To illustrate, the following entry will cause the processor to assign the name MASTERTAPE to channel 1 and tape unit 1:

Line	Label	Operation	OPERAND
01	MASTERTAPE	EQU	1 1 , C U
02			

Control Operations

The Four-Tape Autocoder language includes the following four control operations: Origin Control, Litorigin Control, Branch Control and End Control. These operations are orders to the processor which give the programmer control over portions of the assembly process. Specifically, Origin and Litorigin Control operations give the programmer control over the placement of his program in core storage. Branch Control and End Control operations order the processor to produce unconditional branches to locations specified by the programmer. The formats and usages of the four control operations are presented below.

Origin Control

Origin Control statements order the processor to override its automatic assignment of storage and to begin the assignment of succeeding entries at the particular location specified by the programmer. Thus, they enable the programmer to control storage assignments of instructions, constants and data areas. If an Origin Control statement is not the first entry in a source program, the processor will begin the assignment of storage at location 325.

Origin Control statements may be coded as follows:

Line	Label	Operation	OPERAND
01			ADDR (Format 1)
02	ORIGIN	CNTRL	(B1,a,n,k) (Format 2)
03			ADDR,,S (Format 3)
04			

Format 1

This type of Origin Control statement directs the processor to reset its location assignment counter to the particular location (ADDR) specified in the operand, causing the assignment of succeeding entries to begin at this address. ADDR may be an actual or symbolic address. If an actual address is entered, the programmer must be careful that it does not inadvertently result in overlapped portions of the

program. If a symbolic address is entered, it must have appeared as a label earlier in the program sequence. An * address will refer to the current contents of the location assignment counter, which will be the location assigned to the preceding entry plus one.

EXAMPLES

Line	Label	Operation	OPERAND
01	ORIGIN	CNTRL	1000
02			
03			
04	ORIGIN	CNTRL	XYZ+15
05			

The first entry will direct the processor to begin the assignment of succeeding entries at location 1000. The second entry will direct the processor to begin the assignment of succeeding entries at the location that has been assigned to the symbol XYZ plus 15. The symbol XYZ must have appeared previously as a label.

Format 2

The processor maintains a high assignment counter which keeps a record of the highest location to which any source program entry has been assigned. An Origin Control statement with a blank operand directs the processor to begin the assignment of succeeding entries at the contents of the high assignment counter plus 1. This type of statement would normally be used if the programmer wished to overlap variable length areas in storage; it will direct the processor to resume storage assignments immediately following the longest area.

Format 3

An Origin Control statement with an operand address (ADDR) followed by a comma and the letter s resets the processor's location assignment counter to the specified address and orders the processor *not* to alter the contents of its high assignment counter during the processing of succeeding entries. Normally, literals and library subroutines will be stored in locations immediately following the highest location assigned to the source program, i.e., starting at the contents of the high assignment counter plus 1. This type of statement would generally be used to place a group of instructions in the upper part of storage without altering the contents of the high assignment counter, thereby preventing the possible assignment of literals and/or library subroutines outside the upper limits of storage. One of the other two types of Origin Control statements must follow these instructions in order to reset the location assignment counter and to enable the high assignment counter to function again.

If the first entry in the source program (except for comments cards) is an Origin Control statement in Format 3, the high assignment counter will be maintained at its starting point, i.e., 0000. If the next Origin Control statement in the source program has a blank operand (Format 2), the location assignment counter will be set to 0000. Setting of the location assignment counter to 0000 is not usually desired; therefore, an Origin Control statement with an address in its operand (Format 1) will usually be required under these conditions.

When the IBM 7070 Input/Output Control System is used in the source program, the DIOCS entry will affect the high assignment counter. Regardless of the format of the last Origin Control statement, if any, that precedes the DIOCS entry, the high assignment counter will be set to the location immediately following the one assigned to the last instruction resulting from the DIOCS entry. The setting of the high assignment counter by the DIOCS entry must be considered to avoid unintentional overlap of areas if other Origin Control statements follow the DIOCS entry.

Litorigin Control

As mentioned above, literals and library subroutines will normally be assigned locations immediately following the highest location assigned to the source program. If the programmer wishes to specify the placement of literals and library subroutines that have appeared in the preceding Litorigin Segment (that portion of the source program since either the last Litorigin entry or the beginning of the program, if there is no previous Litorigin entry) will be inserted into the program at the point where the Litorigin entry appears. A Litorigin entry may have one of the following formats:

Line	Label	Operation	OPERAND
01			ADDR (Format 1)
02	LITORIGIN	CNTRL	(Blank) (Format 2)
03			ADDR,,S (Format 3)
04			

The label and operation must be as shown.

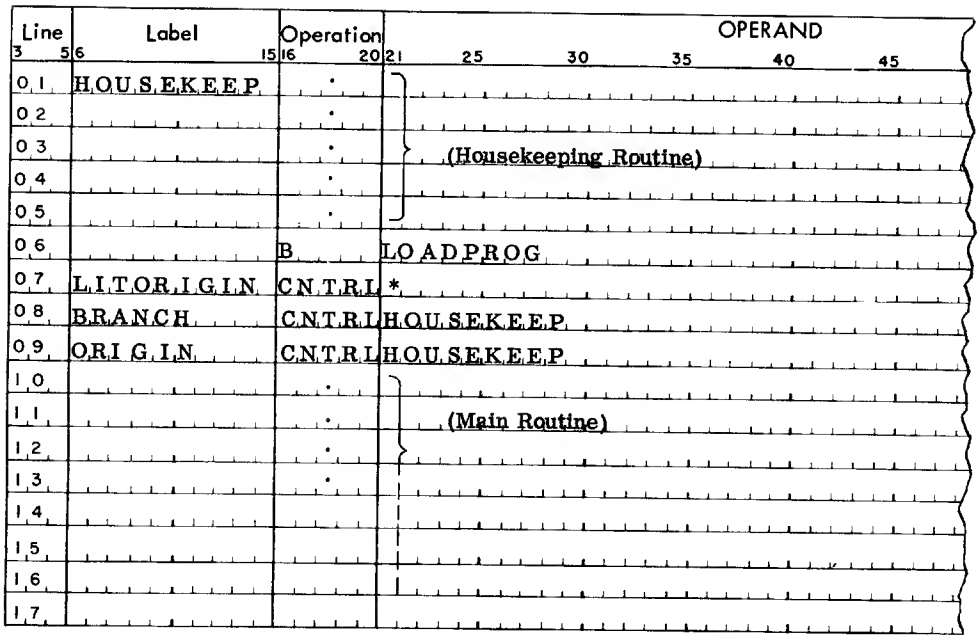
The function of the operand in Litorigin Control is identical to the function of the operand in Origin Control (page 23).

The assignment of storage locations for literals and library routines are made in Phase 4 while all other storage assignments are made in Phase 2. This method of assigning locations requires that an Origin Control entry be used following each Litorigin entry so that the entries following the Litorigin Control statement are not assigned the same locations as the literals; control operations, e.g., BRANCH CNTRL, are the only type of entries which may appear between the Litorigin Control statement and the following Origin Control statement. The Origin Control entry may be omitted wherever the Litorigin Control statement is either the last entry in the source program or followed by an End Control operation. The format of the Origin Control entry which follows a Litorigin Control statement depends on the operand of the Litorigin entry as follows:

If the operand contains an address only (see Line 01), the setting of the high assignment counter is affected and the Origin Control statement must be the type described as Format 1 under "Origin Control."

If the operand contains an address followed by a comma and an s (see Line 02), the setting of the high assignment counter is not affected so the Origin Control entry may be any of the formats described under "Origin Control."

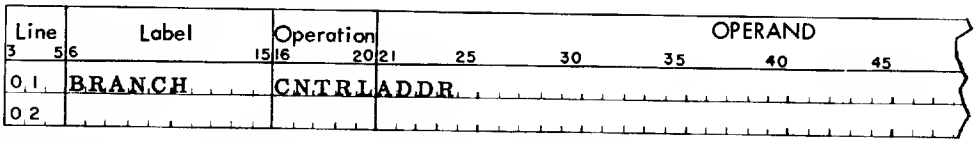
To illustrate the use of a Litorigin Control operation, assume that it is desired to overlap the housekeeping portion of a program with the main routine after the housekeeping routine has been executed. This may be programmed as follows:



The Branch Control entry (Line 08) causes the housekeeping routine to be executed when the program is loaded at object time. The Litorigin Control entry (Line 07) is placed ahead of the Branch Control entry so that the literals and/or library subroutines used in the housekeeping routine will be loaded before the housekeeping routine is executed. The last entry (Line 06) in the housekeeping routine is a Branch instruction which returns control to the load program and causes the main routine to be loaded. (If the IBM 7070 Condensed Card Load Program is being used as recommended, the last instruction in the housekeeping routine may be a Branch to 0000.) The Origin Control entry (Line 09) specifies that the main routine is to be loaded into the area used by the housekeeping routine, i.e., the main routine will be stored beginning with symbolic location HOUSEKEEP.

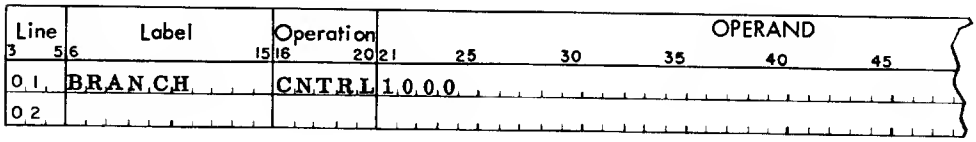
Branch Control

The Branch Control operation



will cause the processor to produce an unconditional Branch instruction which, when encountered during the loading of the object (machine-language) program, will cause the normal loading process to stop and a branch to be executed to the location (ADDR) specified in the operand. ADDR may be actual or symbolic.

For example, the entry



will cause an unconditional branch to location 1000 during the loading of the object program.

A Branch Control entry may be used in conjunction with an Origin Control entry to execute portions of a program already loaded into storage and to overlap these with other instructions. For example,

Line	Label	Operation	OPERAND
01	START	(First instruction)	
02			
03			
04	XYZ	B	LOADPROG
05	BRANCH	CNTRL	START
06	ORIGIN	CNTRL	START
07			
08			
09			

The preceding entries will cause an unconditional branch to the location assigned to the symbol START, followed by the execution of instructions from START through the instruction located at XYZ. The instruction located at XYZ will cause a branch to the load program which will resume the loading of the remainder of the object program. The latter has been assigned locations beginning with the location START and will thus overlap the instructions which have been executed. (In this example, it is assumed that the starting location of the load program is symbolic location LOADPROG.)

End Control

The End Control operation

Line	Label	Operation	OPERAND
01	END	CNTRL	ADDR
02			

is used to begin the execution of the object program immediately after the object program, including literals, has been loaded into storage. It will cause the processor to write out the literals used in the program and to generate an entry containing an unconditional branch to the location (ADDR) specified in the operand. (A Branch Control operation would not be used for this purpose, because the branch would occur before the literals have been loaded.) ADDR may be actual or symbolic.

If an End Control operation is used, it must be the last entry in the source program. If it is not used, the processor will generate an unconditional branch to location 325.

To illustrate, the entry

Line	Label	Operation	OPERAND
01	END	CNTRL	START
02			

will cause an unconditional branch to the location assigned to the symbol START when the loading of the object program, including literals, has been completed.

Part II: 7070/7074 Four-Tape Autocoder Library

Macro-Instructions

In addition to the input and output macro-instructions described in the 7070 Data Processing System Bulletin "IBM 7070 Input/Output Control System," the 7070/7074 Four-Tape Autocoder system provides additional macro-instructions in the Library portion of the System Tape. Each of the macro-instructions in the Library are described separately below. Additional macro-instructions written by the user may be inserted into the Library as explained under "Library Changes."

The macro-instructions processed by Four-Tape Autocoder are substitution-type macros which appear in the Library as short sequences of symbolic language instructions. These sequences of instruction are in "skeleton" form; i.e., they may have missing Labels, Operation Codes and/or Operands. When a macro-instruction is encountered in the source program, the corresponding sequence of skeleton instructions will be obtained from the Library and the parameters in the operand of the original macro-instruction (the one written on the coding sheet) will be inserted into the skeleton instructions. The sequence of "completed" symbolic instructions will then be merged into the source program in place of the original macro-instruction.

The skeleton routine used by a particular macro-instruction is selected from the Library by placing the name of the macro in the Operation column of an entry in the source program. Skeleton routines in the Library are identified by the same name as the macro-instruction which uses it.

Up to nine parameters may be written in the Operand columns of the macro-instruction; the nine parameters must be contained on five lines. When more than one line is needed for a macro, the Label and Operation columns of the additional cards must be blank. The parameters for the macro-instructions described in this manual must be separated by commas and may not contain blanks unless the blanks appear between @ characters. Only input and output macro-instructions may use single blanks with connective words as specified in the publication describing the IBM 7070 Input/Output Control System. Two consecutive blanks in the Operand columns are considered to be the end of the card and are considered as the end of the macro-instruction only when the Operation columns of the next line are *not* blank. Four-Tape Autocoder does not automatically add a comma as the last character of each line of a macro-instruction. Therefore, the last parameter on a line must be followed by a comma, unless it is desired to indicate that a parameter is carried over to the next card. The following examples illustrate the function of commas in the operands.

EXAMPLE 1

No parameters will be omitted if written as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
0 1	ANYLABEL	MNAME	PARA,PARB,PARC,
0 2			PARD,PARE,PARF,PARG,
0 3			PARH,,PARI
0 4			

EXAMPLE 2

Written as follows, the fourth and eighth parameters will be omitted:

Line	Label	Operation	OPERAND
0 1	ANYLABEL	MNAME	PARA,PARB,PARC,,,
0 2			PARE,PARF,PARG,,,
0 3			PARI
0 4			

EXAMPLE 3

Written as follows, the first, fourth, fifth, sixth and eighth parameters will be omitted:

Line	Label	Operation	OPERAND
0 1	ANYLABEL	MNAME	,PARB,,PARC,,,,,
0 2			PARG,,,
0 3			PARI
0 4			

ZSUM

Line	Label	Operation	OPERAND
0 1	ANYLABEL	ZSUM	ACC#,RESULT,A,B,C,D,E,F,G
0 2			

This macro-instruction will set a specified accumulator to zero, add up to seven fields together and store the result.

The accumulator to he used by this macro-instruction is specified by a 1, 2 or 3 as the first parameter (ACC#) in the operand.

After summing the fields specified by the third through ninth parameters, the answer is stored in thc symbolic location specified by the second parameter (RESULT). The address specified may have field definition, address arithmetic, and indexing; when field definition is specified, the unused digit positions of the word receiving the answer will be set to zeros. If the answcr need not be stored, the second parameter (RESULT) may be omitted and the answer will be located in the accumulator specified by the first parameter (ACC#).

The fields to be added together (the third through ninth parameters) may have field definition, address arithmetic, and indexing.

Each ZSUM macro-instruction must specify the first and third parameters; other parameters need be included only when they are necessary. Note that the ZSUM macro-instruction does *not* check for an accumulator overflow condition; if an overflow condition can occur, the programmer must check for it after the macro has been executed.

CALL

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
0 1	ANYLABEL	CALL	SUBROUTINE
0 2			

This macro-instruction will insert a linkage to a library subroutine into the source program and include that subroutine in the program.

The name of the desired subroutine (up to five characters) must be placed in the operand. At the point in the program where the CALL macro-instruction occurs, 7070/7074 Four-Tape Autocoder will insert a linkage to the desired subroutine. At object time, index word 94 is used to hold the address to which control will return after the subroutine has been executed. The CALL macro-instruction also locates the subroutine in the Library and enters it into the program; the subroutine will be stored as indicated by a Litorigin Control entry (see page 25).

If more than one CALL macro-instruction names a certain subroutine, the subroutine will be entered into the program only once in each Litorigin Segment in which it is used. The linkages inserted by each CALL macro-instruction will send control to the start of the subroutine. Return to the proper point in the program, i.e., the instruction following the CALL entry, will be controlled by index word 94.

Writing Substitution-type Macro-instructions

Additional macro-instructions may be included in Four-Tape Autocoder by writing skeleton routines and inserting them into the Library as explained under "Library Changes." The five-character name of a macro is assigned by a Library Change Card at the time the macro-instruction is placed into the Library.

Each skeleton instruction, which forms part of a substitution-type macro-instruction in the Library, may consist of an Autocoder entry in various stages of completion as follows:

The instruction may be complete and will be used each time the macro is called for.

The instruction may be partially complete and require that the omitted parameter(s) be supplied each time the macro is used. When this type of skeleton instruction is used and a required parameter(s) is omitted, the skeleton instruction will appear in the routine without the missing parameter(s) and an error message to indicate the omission will be typed.

The instruction may be partially complete and used only when the omitted parameter(s) is supplied by the macro. If the macro does not supply the required parameter(s) for this type of skeleton instruction, the skeleton instruction will be deleted from the routine; no message will result from this deletion.

END CNTRL or LITORIGIN CNTRL cannot be used in skeleton routines or subroutines.

Any portion of an Autocoder entry, except the label, may be complete in a skeleton instruction. Labels for skeleton instructions may be specified if the macro-instruction can occur only once in any one program; use of the macro more than once in the source program would produce duplicate symbols and result in improper assembly of the program.

A special two-character code is used within skeleton instructions to indicate that a source program macro-instruction parameter is to be inserted, or may be in-

serted, into the skeleton instruction in place of the code. The first character is a ¤ character (lozenge) which identifies the code and causes the insertion to be initiated. The second character identifies which parameter is to be inserted and also indicates whether the parameter must be present or may be omitted. The special two-character code may also be used in place of symbolic labels within the skeleton routine.

A parameter which must be present each time the macro is used must be identified by ¤1 through ¤9 in the skeleton instruction. The number following the ¤ character specifies the position of the parameter in the operand of the macro-instruction; thus, ¤4 specifies the fourth parameter of the macro-instruction.

Parameters in skeleton instructions which are to be used only when the parameter appears in the operand of the macro must be identified by ¤A through ¤I. The numerical portion of the character following the ¤ character specifies the position of the parameter in the operand of the macro-instruction, e.g., I is the ninth parameter.

The label, if any, for a substitution-type macro is indicated by ¤0 in the Label column of a skeleton instruction. The label for the skeleton routine will be the same as the label of the macro-instruction that supplies the parameters. If the ¤0 appears in the label of a skeleton instruction that may be deleted by omitting its parameter, the label will be assigned to the first subsequent skeleton instruction that is used.

To illustrate the writing of a substitution-type macro-instruction, the skeleton routine of the ZSUM macro (see page 30) will be used; it appears in the Library as follows:

Line	Label	Operation	OPERAND
0 1	¤0	Z A ¤1	¤3
0 2		A ¤1	¤D
0 3		A ¤1	¤E
0 4		A ¤1	¤F
0 5		A ¤1	¤G
0 6		A ¤1	¤H
0 7		A ¤1	¤I
0 8		Z S T ¤1	¤B
0 9			

As explained in the description of the ZSUM macro-instruction, operands 1 and 3 must always be supplied; this is specified in the skeleton instructions by the use of ¤1 and ¤3. Other skeleton instructions are optional, so the parameters are specified by ¤B and ¤D through ¤I. The ¤0 in the label indicates that a label may be assigned if desired.

For example, if the following ZSUM macro-instruction is written into a program,

Line	Label	Operation	OPERAND
0 1	CALCGROSS	Z SUM	3 , GROSSPAY , SALARY , OVERTIME ,
0 2			COMMISION
0 3			

the following sequence of instructions will be inserted into the source program:

Line	Label	Operation	OPERAND
01	CALCGROSS	ZA 3	SALARY
02		A 3	OVERTIME
03		A 3	COMMISION
04		ZST 3	GROSSPAY
05			

The sixth through ninth parameters of the macro-instruction are omitted by not entering them on the coding sheet; no message will be typed during assembly because ¤F through ¤I in the operands of the skeleton instructions indicate that the parameters may be omitted. Similarly, if the second parameter (GROSSPAY) had been omitted, the ZST3 instruction would not appear in the source program nor would a message be typed.

The programmer may write the operand of a skeleton instruction so that a choice of parameters from the macro will be made. A choice of parameters is specified in a skeleton instruction by separating parameters in the operand with a / character (slash). The parameter preceding the / character must be identified by ¤1 through ¤9; the parameter following the / character may be identified by ¤1 through ¤9, ¤A through ¤I or it may be a literal.

If the parameter preceding the / character appears in the operand of the macro-instruction, the parameter from the macro will be used and the parameter following the / character in the skeleton instruction will be ignored. If the parameter preceding the / character is omitted from the macro, the parameter following the / character will be used. When both parameters are omitted from the macro, the processing of the skeleton instruction will depend on the form of the parameter that follows the / character. The form of the parameter affects the processing as follows:

> If the parameter following the / character is identified by ¤1 through ¤9, the skeleton instruction will appear in the source program without the omitted parameter; i.e., the skeleton instruction will be made as complete as possible and placed in the program. An error message indicating the omission will be typed.
>
> If the parameter following the / character is identified by ¤A through ¤I, the skeleton instruction will be omitted completely; no error message will occur.

The following skeleton instructions illustrate various choices of parameters:

Line	Label	Operation	OPERAND
01		ZA 1	¤1/+1000
02		STD 1	¤4/¤2
03		B	¤3/¤E
04			

The ZA1 instruction will always appear in the source program; +1000 will be used whenever the first parameter is omitted from the macro-instruction. The STD1 instruction will use the second parameter whenever the fourth parameter is omitted from the macro; if neither parameter appears in the macro-instruction, the STD1 instruction will appear in the source program without an operand and an error message will be typed. The Branch instruction will use the fifth parameter whenever the third parameter is omitted from the macro-instruction; if neither parameter

appears in the macro, the Branch instruction will not appear in the source program and no message will be typed.

The parameters in a macro-instruction and the operands in a skeleton instruction may have field definition, address arithmetic, and indexing. Use of these programming methods will be interpreted as follows:

If field definition is used in both the skeleton instruction and the parameter supplied by the macro, the field definition in the skeleton instruction will be considered as relative to the field definition in the parameter. For example, if the field definition in the operand of the skeleton instruction is (2, 4) and that in the parameter of the macro is (2, 7), the field definition of the instruction inserted into the source program will be (4, 6).

If address arithmetic is used in both the skeleton instruction and the parameter supplied by the macro, the instruction inserted into the source program will have address arithmetic equal to the algebraic sum of the two. For example, if the address arithmetic of one is +7 and the other is −4, the instruction in the source program will have address arithmetic equal to +3.

Indexing may be used in either the parameter supplied by the macro or the operand of the skeleton instruction but not both. If a parameter contains an actual index word that is used only for indexing, the index word must be written in the form xnn where nn is an index word number from 1 through 99. The index word, in the form xnn, from the parameter may not be used as the first item in the operand of an index instruction, e.g., an Index Word Load (XL) instruction, because xnn will be interpreted as a symbolic index word as explained under "Uses Other Than Indexing."

To illustrate field definition, address arithmetic, and indexing, assume that parameter 6 of a macro is WORKAREA (1, 7) +23+WORKIW and the operand of one of the skeleton instructions which uses that parameter is ¤6 (0, 3)−1. The operand of the instruction inserted into the source program would be WORKAREA(1, 4) + 22 +WORKIW.

In order to allow symbolic references to other skeleton instructions within a routine, symbolic addresses may be generated automatically through the use of ¤J through ¤R in skeleton instructions. To properly generate a symbolic address the symbol, e.g., ¤P, must be used as a label of one skeleton instruction and in the operand of at least one other skeleton instruction in the same skeleton routine. Skeleton instructions that have a label of ¤J through ¤R may *not* have ¤A through ¤I as operands; if such a combination were used and a skeleton instruction were deleted because a parameter is omitted from the macro, the instruction referring to the deleted instruction would have an undefined symbol in its operand.

To avoid duplicate labels, they are generated in the form MACROmmmmn where mmmm is the number of the macro within the source program and n is the numerical portion of the character identifying the symbolic address, i.e., 1 through 9 for J through R respectively. An example of a generated label appears in the coding example below.

To illustrate symbolic addressing within a skeleton routine, assume that the Library contains the following skeleton routine:

Line	Label	Operation	OPERAND
01		.	
02		.	
03		B	¤J
04		.	
05		.	
06	¤J	ZA 1	¤1
07		.	
08		.	
09			

Assume that the macro-instruction supplied the name COST for the first parameter and that this is the twenty-fifth macro in the source program. The following instructions will be inserted into the source program:

Line	Label	Operation	OPERAND
01		.	
02		.	
03		B	MACRO 0 0 2 5 1
04		.	
05		.	
06	MACRO 0 0 2 5 1	ZA 1	COST
07		.	
08		.	
09			

Operands of instructions used in skeleton routines may use the special symbol * with address arithmetic only if all instructions between the one with the * symbol and the one referred to will be present always, i.e., the operands do not specify optional parameters by using ¤A through ¤I. Two examples of the proper use of the special symbol * follow.

EXAMPLE 1

Use of the * symbol to branch around constants.

Line	Label	Operation	OPERAND
01		.	
02		.	
03		B	*+3
04		DC	
05	¤J		+1 4 4
06	¤K		+1,2,3,4,5,(0,,4)
07	¤L		+2,5,(8,,9)
08		A ¤1	¤B
09		.	
10		.	
11			

EXAMPLE 2

Use of the * symbol to form a loop.

Line	Label	Operation	OPERAND
01		.	
02		.	
03		ZA ⌑1	⌑2+⌑4
04		A ⌑1	⌑3+⌑4
05		SRR ⌑1	3
06		ZST ⌑1	⌑5+⌑4
07		BIX	⌑4,*-4
08		.	
09		.	
10			

Operands of instructions used in skeleton routines may use literals as required. However, it is not possible to substitute parameters into an alphameric literal or constant because @⌑N@ is assumed to be the desired coding. For example, if the following coding appeared in a skeleton routine,

Line	Label	Operation	OPERAND
01		.	
02		.	
03		B	*+3
04	⌑J	DC	-RDW
05			@⌑5@
06		TYP	⌑J
07		.	
08		.	
09			

execution of the routine will cause the two characters ⌑5 to be typed regardless of what appeared as the fifth parameter of the macro-instruction which uses this routine.

None of the instructions used in the skeleton routine of one macro-instruction may be another macro-instruction. However, it may be desirable to use a library subroutine at some point in a macro-instruction so the special operation INCL has been made available in Four-Tape Autocoder. The special operation INCL used with a BLX instruction in a skeleton routine performs the same function as a CALL macro in the source program. The following entries must be made to use a subroutine in the skeleton routine of a macro-instruction:

Line	Label	Operation	OPERAND
01	ANYLABEL $	BLX	9,4,SNAME,$
02		INCL	SNAME
03			

Special operation INCL may appear only in the skeleton routine of a macro-instruction or in a subroutine. The above example assumes that the subroutine (SNAME) has been written to be called in by the CALL macro as explained under "Subroutines." (The use of the dollar sign in this example is explained on page 38.)

EXAMPLE 3

Assume that a macro-instruction to compute the value of $\sqrt{x+z}$ is to be written.

If the Library contains a subroutine for calculating the square root of a number, the subroutine may be included in the skeleton routine; for this example, assume that the subroutine is named SQRT and that it takes the square root of a number placed in accumulator 1 and places the answer in accumulator 3. The skeleton routine for this operation could be written as follows:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
01	¤0	ZA,1	¤1
02		A,1	¤2
03		BLX	94,SQRT.$
04		INCL	SQRT
05		Z,ST,3	¤3
06			

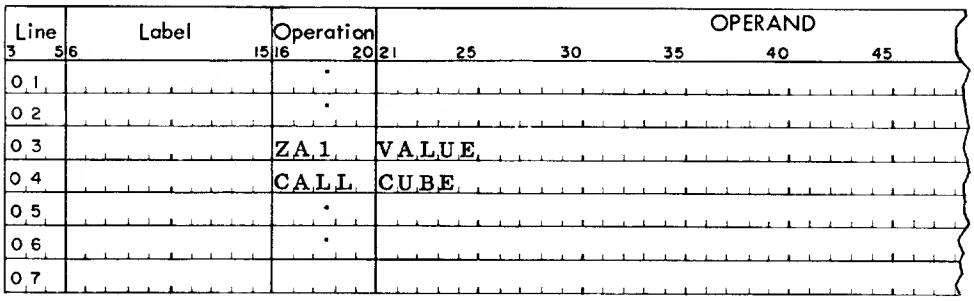
The macro-instruction which uses this skeleton routine would specify the location of x as the first parameter, the location of z as the second parameter and the third parameter would specify the location into which the answer is to be stored.

Subroutines

The 7070/7074 Four-Tape Autocoder system allows subroutines to be added to the Library portion of the System Tape. Library subroutines are commonly used sequences of instructions which can be loaded into one area of storage and entered from any number of points in the source program. A subroutine differs from a macro-instruction in the following ways:

1. All instructions in a subroutine are complete; no parameters may be inserted.
2. Subroutines usually consist of many instructions which would occupy too many storage locations if the whole sequence of instructions were repeated each time the subroutine were used. Although a subroutine entry may be entered from many points in a program, each subroutine will appear only once within the Litorigin Segment in which it is used.
3. Subroutines are not included in the program at the point where the CALL macro is used; the locations into which the subroutine is stored will be determined by a Litorigin entry (see page 25).
4. Data to be operated on by the subroutine must be placed in specific locations as required by the individual subroutine.

Subroutines are incorporated into a source program by using the name of the subroutine in a CALL macro-instruction (see page 31). Before the CALL macro is used, all data required by the subroutine must be placed in the storage location(s) where the subroutine expects to find it. For example, if a library subroutine named CUBE for raising a number to the third power requires that the number to be cubed be in accumulator 1 at the start of the subroutine, the number (VALUE) must be placed in accumulator 1 before the CUBE subroutine is called for; coding to do this would be as follows:



The result of the subroutine would always be placed in the storage location specified by the subroutine. In the example above, the subroutine might be written to store the result in accumulator 1 in place of the original number.

Writing Subroutines

The proccdure for inserting subroutines into the Library is explained under "Library Changes." The name of a subroutine may contain up to five characters and is assigned by a Library Change Card at the time the subroutine is inserted into the Library.

The ability to use a library subroutine in any program processed by 7070 Four-Tape Autocoder requires that care be taken to avoid accidental duplication of symbols that are used in a subroutine and a source program. Duplicate symbols can be avoided by using a period (12-8-3 punch) within the symbols contained in a subroutinc beeause periods may not be used in symbols of source programs.

The function of the Litorigin entry in storing literals and library subroutines could introduce duplicate symbols if a subroutine were used in more than one section of a source program, i.e., a subroutine is named in CALL macro-instructions that are separated by a Litorigin entry. To avoid duplicate symbols written within subroutines, a dollar sign (11-8-3 punch) may be used within the symbols appearing in the subroutine. Four-Tape Autocoder will substitute a diffcrent alphabetic character for the dollar sign in each section of the program. For example, if a label in a subroutine were CALCTOT.$, the label would appear as CALCTOT.A for the first Litorigin, as CALCTOT.B for the second Litorigin, etc.; if the subroutine were called for only in the first and fourth sections of a program, the label would appear as CALCTOT.A for the first time and CALCTOT.D the next time.

If a subroutine is to be called for by the CALL macro-instruction provided with 7070/7074 Four-Tape Autocoder (rather than a similar macro written by the user), index word 94 must be used to return control to the proper point in the source program after the subroutine has been executed. This is easily done by executing a Branch to 0+x94 when processing in the subroutine has been completed.

When writing subroutines to be inserted into the Library, any Autocoder entries that are acceptable to Four-Tape Autocoder may be used except that subroutines may not include any macro-instructions. However, it may be desirable to use one subroutine at some point in another subroutine so the special operation INCL has been made available in Four-Tape Autocoder. The special operation INCL used with a BLX instruction in a subroutine performs the same function as a CALL macro in the source program. The following entries must be made to use one subroutine in another:

Line	Label	Operation	OPERAND
01	ANYLABEL	BLX	94,SNAME,$
02		INCL	SNAME
03			

The special operation INCL may appear only in a subroutine or the skeleton routine of a substitution-type macro-instruction. The above example assumes that the subroutine (SNAME) has been written to be called in by the CALL macro, i.e., return to the proper point in the program is controlled by index word 94.

EXAMPLE

Assume that a subroutine to solve the following formula and store the answer in accumulator 1 is to be written:

$$\text{FREQ} = \frac{10^6}{2\pi\sqrt{LC}}$$

If the Library contains a subroutine to find the square root of a number placed in accumulator 3, the square root subroutine (SQRT) may be used to find the value of $\sqrt{LC}$ as follows:

Line	Label	Operation	OPERAND
01	FREQ,$		
02			Calculate value of L times C and
03			store value in PRODLC,$
04		ZA 3	PRODLC,$
05		XL	XSUBREXIT,94 Save contents of I.W. 94
06		BLX	94,SQRT,$
07		INCL	SQRT Assume result is stored in ROOT,$
08			
09			Calculate $\frac{10^6}{2\pi\sqrt{ROOT}}$,$ and store
10			answer in accumulator 1
11			
12		B	0,+XSUBREXIT
13			

Part III: 7070/7074 Four-Tape Autocoder Processor

Organization of the Processor

Program Functioning

The Four-Tape Autocoder processor is divided into six sections named: System Control, Librarian, Phase 1, Phase 2, Phase 3 and Phase 4. The first section is used to control the other sections; the Librarian is used to add to, delete from, change and/or duplicate the System Tape. Phases 1 through 4 are used to convert the source program which is on cards and/or tape to a machine-language program. Two tapes are produced by the processor; one contains the assembled machine-language program in condensed card format and the other contains an edited listing showing the symbolic-language source program entries and the corresponding assembled machine-language instructions. Additional on-line printing and/or punching of the output may be elected through the use of control cards (see "Option Cards").

The operation of the Four-Tape Autocoder processor is shown in the general flow chart in Figure 2. The functions performed in each section of the processor are explained briefly below. Programmers who are interested in the methods used to perform these functions are referred to Appendix A which contains a general flow chart of each phase.

System Control

This section of the processor is loaded into storage from the System Tape and remains there throughout Autocoder assembly to control the operation of each phase of the processor. (The Librarian section is loaded into storage with the System Control section; however, it may or may not remain in storage depending on the type of run as explained below.)

System Control first reads any option cards (see page 49) which may be necessary to alter one or more of the "permanent" options for a particular machine run. From a RUN Control Card, the System Control section will determine whether an original assembly, a re-assembly or a system run is to be done. If the RUN Control Card indicates a system run, System Control will cause the Librarian section of the processor to be executed. If either an original assembly or a re-assembly run is indicated, a table contained in the Librarian section will be preserved in storage and Phase 1 will be loaded into the storage locations occupied by the Librarian section; the table that is saved contains the names of all subroutines and macro-instructions which are available from the Library portion of the System Tape. After Phase 1 has been loaded, it is executed. As each phase is completed, System Control loads the next phase and causes it to be executed.

Librarian

The Librarian section of the processor is executed only when a system run is made. In a system run, the Librarian section provides for adding to, deleting from, and changing the System Tape; it may also be used to produce a duplicate of the System Tape.

As stated above, the Librarian and System Control are loaded into storage together although only the table of macro-instructions and subroutines required in Phase 1 is retained when an original assembly or re-assembly run is made.

Phase 1

The System Control section causes Phase 1 to be loaded and executed whenever the RUN Control Card specifies either an original assembly or re-assembly run. The operations performed in Phase 1 are as follows:

1. The source program is read from tape and/or cards.

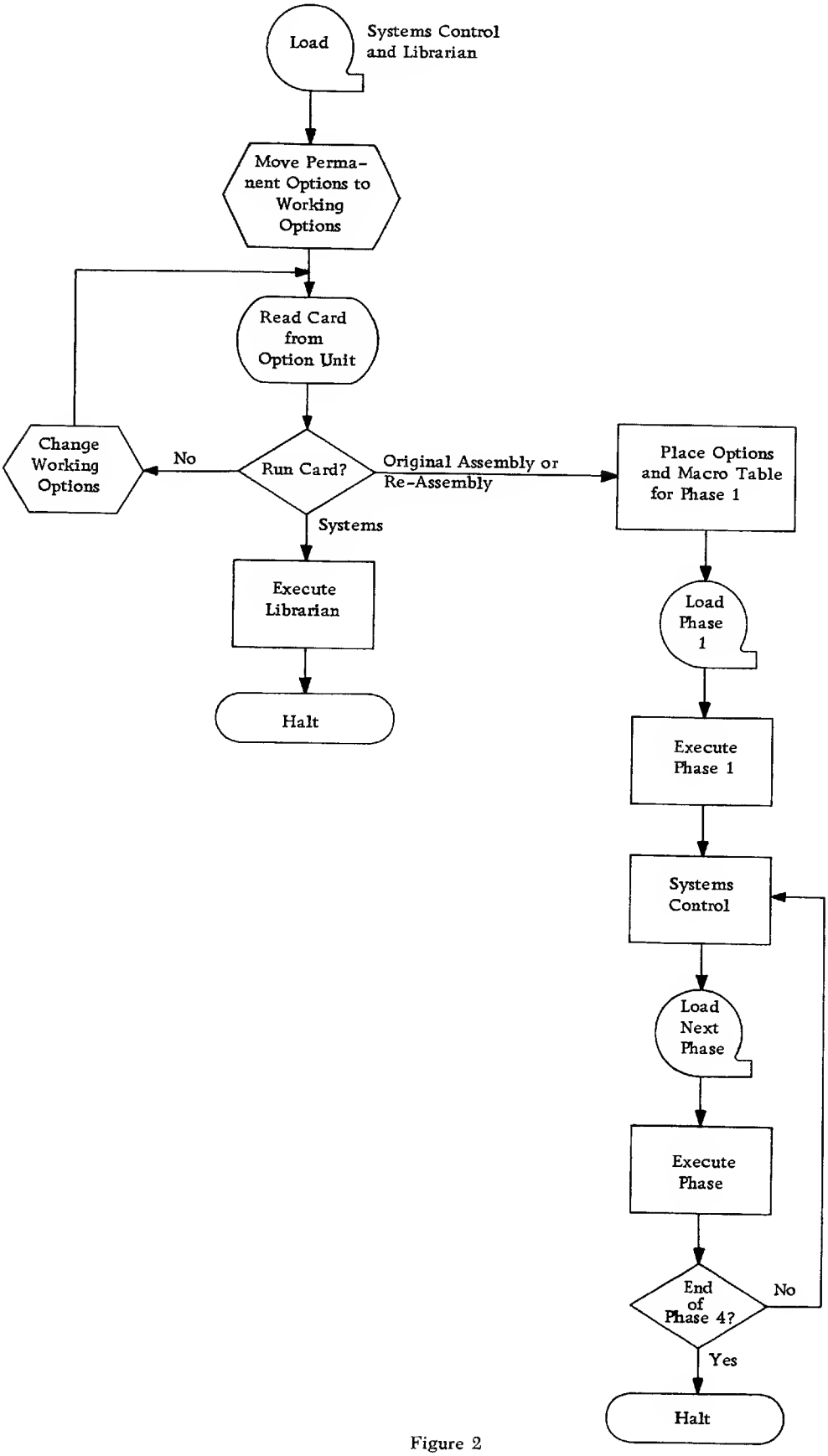

Load
Systems Control and Librarian
Move Permanent Options to Working Options
Read Card from Option Unit
Change Working Options
No
Run Card?
Original Assembly or Re-Assembly
Place Options and Macro Table for Phase 1
Systems
Execute Librarian
Halt
Load Phase 1
Execute Phase 1
Systems Control
Load Next Phase
Execute Phase
End of Phase 4?
No
Yes
Halt

Figure 2

2. Page and line numbers are checked for ascending sequence.
3. The skeleton routine corresponding to each macro-instruction used in the source program is obtained from the Library and inserted in sequence at the point in the source program where the macro-instruction was used.
4. A linkage to a subroutine is inserted at the point in the source program where a CALL macro-instruction names that subroutine. The instructions which form the subroutine will be obtained from the Library and entered into the source program at a point specified by a Litorigin Control entry.
5. LITORIGIN CNTRL entries are partially processed in this section of the processor. Each Litorigin Control entry produces an Origin Control entry which is placed immediately ahead of the Litorigin Control entry. If any subroutines are used in the program, they will be inserted between the generated Origin Control entry and the source program Litorigin Control entry.
6. Declarative entries which require symbolic operands to be defined earlier in the program sequence, e.g., DRDW with a single operand and EQU, are placed in a table for use during Phase 2.
7. A 17-word output tape record to be used as input to Phase 2 is written for each entry in the source program. Macro-instructions and subroutines are exceptions to this procedure; there will be a 17-word output tape record for each instruction contained in a subroutine or macro-instruction.
8. Messages will be prepared and typed for errors which are detected by this phase of the processor.

Phase 2

The output records produced by Phase 1 are processed by Phase 2, which performs the following functions:

1. Each operation code is checked to see if it is a valid Autocoder operation code. Invalid operation codes will be processed as NOP instructions.
2. Imperative Autocoder entries are converted to the corresponding 7070 machine-language codes.
3. A table of symbolic labels and the corresponding storage locations assigned during this section of the processor is generated. If the number of symbols fills the storage area allotted to the symbol table, the table will be written on tape and a continuation of the table will use the same storage area. (This method of processing avoids limiting the number of symbols that may appear in a source program.)
4. Storage locations are assigned to fields and constants defined by DA and DC entries.
5. The operands of imperative Autocoder entries are processed as completely as possible. Operands containing symbolic addresses cannot be processed completely at this time because the symbol table is generated during Phase 2 and is not available until the end of this phase. Actual addresses, asterisks, index words, electronic switches, etc. which appear in operands are processed during this phase.
6. Availability tables which indicate actual index words and electronic switches used in the source program are generated; these tables are used in Phase 3 to avoid duplication when assigning symbolic index words and electronic switches.
7. A 26-word output tape record to be used as input to Phase 3 is written for each Phase 2 input record. The 26-word record contains, in addition to the

input information, symbols which appear in operands plus an indication of how the symbol is used. For example: is the symbol the name of an index word; is it a literal rather than a symbol; does the symbol have address arithmetic?

8. Messages will be prepared and typed for errors which are detected by this phase of the processor.

Phase 3

Processing of the output records produced by Phase 2 may require one or more passes of Phase 3. The number of passes depends on the number of symbols used in the program; if all symbols can be contained in the allotted storage area at one time only one pass is required. The functions performed by this section of the processor are as follows:

1. A check for duplicate labels assigned to different locations is made and an error message will be typed each time this condition is detected.
2. The storage location assigned to each symbolic address is inserted into the partially completed instructions produced by Phase 2.
3. During the last pass of Phase 3, which may be the first pass if only one is required, all symbolic index words and electronic switches are reserved and assigned to available index words and electronic switches; the availability tables generated in Phase 2 are used to avoid duplicate assignments. Symbolic index words and electronic switches are placed in a special table for use in Phase 4.
4. The last pass of Phase 3 will also insert the assigned index word or electronic switch into instructions (except arithmetic type) that define a symbol as an index word or electronic switch by its use in the operand rather than by an EQU entry.
5. An output record to be used as input for Phase 4 is written on tape.
6. Messages to indicate errors detected during this phase of the processor will be prepared and typed.

Phase 4

This phase is the last section in Four-Tape Autocoder and its output is the object program. The following functions are performed in this section of the processor:

1. Arithmetic type instructions containing symbolic index words and/or electronic switches are completed during this phase by inserting the assigned index words and electronic switches. The special table of symbolic names and the corresponding assigned index words and electronic switches prepared in Phase 3 is used for this purpose.
2. All literals are "packed" into storage locations. These assigned locations with appropriate field definition are inserted into the instructions which use the literals.
3. All instructions are checked for completeness; a message indicating an undefined symbol will be typed for incomplete instructions.
4. A tape containing the object program in IBM 7070 Condensed Card format is written; a typed message will indicate which tape unit holds this output file. This tape is written in the alphameric mode for punching into cards by an off-line (or SPOOL) tape-to-card operation; cards punched from this tape by an off-line tape-to-card operation will contain a load card indicator in both columns 65 and 79. (The IBM 7070/7074 Condensed Card Load Program is automatically included as the first five cards of each program assembled using Four-Tape Autocoder.)

5. An edited listing tape showing the symbolic-language source program entries and the corresponding assembled machine-language instructions is written; the listing includes a table of all index words and electronic switches that are *not* used by the object program. A typed message will indicate which tape unit holds this output file. This tape file may be printed off-line to obtain a listing which shows each entry of the source program and, on the same line, the corresponding machine-language instruction; the assigned location of the instruction also appears on the line. One use of the listing is in preparing "patch" cards to correct or alter an assembled program. The edited listing tape may also be used as an input tape for re-assembly runs of the Four-Tape Autocoder processor.

6. If any on-line printing and/or punching options are elected (see "Option Cards"), the on-line operation(s) will occur simultaneously with the writing of the two output tapes.

Types of Runs

The functions performed in each of the three possible types of runs are described below, together with the form of input required for each run.

Original Assembly Run

An original assembly run processes a source program in Autocoder form into a machine-language program. The input to an original assembly run may consist of either a deck of cards containing the Autocoder statements or a reel of tape prepared from those cards. When the source program is on tape, each tape record must be one Autocoder statement written in the alphameric mode, i.e., a mode change character may not be the first character of the tape record.

Re-assembly Run

A re-assembly run makes additions, changes and/or deletions to a previously assembled program. Re-assembly runs provide the means to correct errors that have been noted during either the testing of assembled programs or previous assemblies. Two forms of input are used for a re-assembly run; one is the edited listing tape that was produced by an original assembly or a previous re-assembly, the other is Autocoder statements giving additions and changes or indicating deletions. The additions, changes and deletions may be read directly from cards or the cards may be written on tape by a card-to-tape operation and the tape used as input. The additions, changes and deletions are merged with the original source program as it is listed to produce a new source program which includes the corrections. The new source program is processed as though it were the input for an original assembly. At least one addition, change or deletion must be included for each re-assembly run; if for some reason it is desired to re-assemble a program with no change in logic, this requirement may be met by changing one entry to itself.

Both the source program and the Autocoder statements for additions, changes and deletions must be in ascending sequence according to the page and line numbers. The page and line numbers are used by the processor to locate the statements to be affected by the re-assembly run.

Additions and Changes

To change an Autocoder entry in the program being re-assembled, a new Autocoder entry must be prepared using the page number and line number of the one to be changed. The processor will locate the entry to be changed on the edited listing tape by comparing page and line numbers; when an old entry equal to the new entry is found, the new Autocoder entry will replace the old entry. If the edited listing tape does not have an entry with page and line numbers equal to a new entry, the new Autocoder entry will be regarded as an addition and inserted into the program in proper page and line number sequence.

DELETIONS

Autocoder entries in the program being re-assembled may be deleted through the use of an Autocoder statement having DELET in the Operation columns. One entry or several consecutive entries may be deleted depending upon the operand of the DELET statement. Depending on the number of entries to be deleted, one of the following forms of DELET statement must be used:

Page & Line No.	Label	Operation	Operand
FI RS T		DELET	FINAL
ONENO		DELET	

The first line shows the form of a DELET statement to be used to delete any number of consecutive Autocoder entries from the program being re-assembled. Columns 1 through 5 of the DELET statement must contain the page number and line number of the first entry to be deleted; the operand must contain the page and line number of the last entry of the sequence that is to be deleted.

The second line shows the form of a DELET statement to be used to delete one Autocoder entry from the program being re-assembled. The page number and line number of the entry to be deleted must be placed into columns 1 through 5 of the DELET statement; the operand must be blank.

Note that a DELET statement is not required when a new Autocoder entry is to replace an old one; replacing one entry with a new entry is explained above under "Additions and Changes."

EXAMPLES

The following examples illustrate the use of DELET statements:

Page & Line No.	Label	Operation	Operand
MX76		DELET	34A91
9 8 J34		DELET	

The first line will delete all entries from MX76 through 34A91; the second line will delete entry 98J34.

System Run

A system run makes changes to "permanent" options, Library routines, and/or program portions of the Four-Tape Autocoder processor. Three forms of input are used for a system run; one is the System Tape to be changed, another is Autocoder statements for changes of "permanent" options, subroutines and macro-instruction skeleton routines, the other form is condensed load cards for changes of program portions of the Four-Tape Autocoder processor. The changes may be read directly from cards or the cards may be written on tape by a card-to-tape operation and the tape used as input. If no changes are provided, a system run will produce a duplicate of the System Tape used as input.

The methods of changing various portions of the System Tape are described separately below. Any number of portions of the System Tape may be changed during one system run.

PERMANENT OPTION CHANGES

To change "permanent" options on the System Tape, Autocoder entries specifying the new "permanent" options must be prepared; see "Option Cards." An ASIGN card must precede the option card(s) which specifies the new "permanent" options.

An ASIGN card is prepared as follows:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
0 1		ASIGN	
0 2			

After the system run, the options specified in the option cards which followed the ASIGN card will be "permanent" options on the System Tape. The new "permanent" options can be replaced by temporary working options for individual runs through the use of option cards ahead of the RUN Control Card of any run.

Library Changes

Subroutines and macro-instruction skeleton routines may be added, changed or deleted from the Library portion of the System Tape through the use of library change cards. When inserting new items into the Library, a library change card also serves to assign a name to the subroutine or macro-instruction being added. Library change cards may be in one of the following forms:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
0 1	MACRONAME	INSERM	,N
0 2	SRNAME	INSERS	,N
0 3	MACRONAME	DELETM	
0 4	SRNAME	DELETS	
0 5	MACRONAME	UPDATM	,N
0 6	SRNAME	UPDATS	,N
0 7			

The label of all forms of library change cards must contain the name (up to five characters) of the macro-instruction or subroutine to be added, replaced, deleted or changed.

The first two lines of the coding sheet show the entries to be used when adding or replacing macro-instructions and subroutines respectively; the Operation columns must always contain INSER. The first character in the operand must be either an M or an S to indicate whether the addition or replacement is a macro-instruction or a subroutine. A number (N) from 1 through 9999 preceded by a comma may be included in the operand to specify the number of cards (not counting the INSER card) which form the subroutine or macro-instruction skeleton routine being added or replaced. Whenever the number is included in the operand, the processor will compare that number against the number of cards actually used for the subroutine or macro-instruction skeleton routine; if the number of cards does not agree with the number in the operand, an error message will be typed.

Whenever additions or replacements are made, the INSER card must be followed immediately by the cards containing the subroutine or skeleton routine named in the label of the INSER card. Subroutines and skeleton routines must be in the cards in Autocoder form. An INSER card for replacing an item in the Library will use the name of a subroutine or macro-instruction that already exists in the Library; the item already in the Library will be deleted before its replacement is inserted.

The next two lines of the coding sheet show the entries to be used for deleting entire macro-instructions and subroutines respectively; the Operation columns must always contain DELET. The operand must be either an M or an S to indicate whether the item to be deleted is a macro-instruction or a subroutine.

The last two lines of the coding sheet show the entries to be used for changing macro-instructions and subroutines already existing in the Library. Again the operand must be an M or an S to identify the item to be changed. A number (N) may be included as in the case of the INSER card. The UPDAT card must be followed immediately by the cards containing the changes. The format of these cards is identical to the cards described under "Additions and Changes" and "Deletions" in a Re-assembly Run on page 41.

PROGRAM CHANGES

Program portions of Four-Tape Autocoder may be changed through the use of UPDAT cards followed by the changes in condensed card format. An UPDAT card is required to specify the portion of the System Tape to be changed. The five forms of UPDAT cards that may be used are as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
0 1	SYSTEMS	UPDATN	
0 2	PHASE1	UPDATN	
0 3	PHASE2	UPDATN	
0 4	PHASE3	UPDATN	
0 5	PHASE4	UPDATN	
0 6			

The label of the UPDAT card indicates the portion of the System Tape to be changed. For a system run, the System Control section and the Librarian section are regarded as one; the UPDAT card to be used when changing the System Control and/or Librarian sections is shown on the first line.

The operand of the UPDAT card may be blank or may contain a number to specify the number of change cards which follow the UPDAT card. When a number is included in the operand, the processor will compare that number against the number of cards that actually follow the UPDAT card and type a message if the two do not agree. Even though the number of change cards differs from the number in the operand of the UPDAT card, all of the change cards will be processed.

Only one of each form of UPDAT card may be included in a system run, i.e., UPDAT cards with duplicate labels must not be used. The UPDAT cards, followed by the changes to the section of the processor named by the label, may be read in any order, e.g., the UPDAT card and changes for Phase 4 may be followed by the UPDAT card and changes for Phase 2.

RUN Control Cards

One RUN Control Card must be provided each time the Four-Tape Autocoder processor is used. The type of RUN Control Card needed depends on the processing that is to be done during a particular run. A RUN Control Card for each of the three possible types is described below.

Original Assembly Card

The RUN Control Card for an original assembly is punched as follows:

Line 3 5	Label 6 15	Operation 16 20	OPERAND 21 25 30 35 40 45
0 1	ORIGINAL	RUN	
0 2			

The operand must be blank.

The source program for an original assembly may consist of a deck of Autocoder cards to be read from an on-line card reader or a tape that was prepared from Autocoder cards by an off-line card-to-tape operation.

Re-assembly Card

The RUN Control Card for a re-assembly run is punched as follows:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
0 1	RECOMPILE	RUN	
0 2			

The operand must be blank.

The source program for a re-assembly run is the edited listing tape from an original assembly or a previous re-assembly. Other input for a re-assembly run will consist of additions, changes and/or deletions to the source program as explained under "Re-assembly."

System Card

The RUN Control Card for a system run is punched as follows:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
0 1	SYSTEMS	RUN	
0 2			

The operand must be blank.

The input for a system run is the current System Tape. Other input for a system run will consist of additions, changes and/or deletions to the Systems Tape. A system run with no additions, changes or deletions will produce a duplicate of the current System Tape.

Option Cards

Option cards may be used to temporarily ignore a "permanent" option written on the System Tape and substitute a different working option for the current program run. Option cards specify working options which select input/output units, tape label procedures, and the size of core storage of the 7070 or 7074 systems to be used for Autocoder assembly and/or for running the object program.

Whenever an option card(s) is used for an original assembly or a re-assembly, it must be loaded immediately ahead of the RUN Control Card. An option card(s) used for a system run may be placed either before or after the RUN Control Card depending on the function intended; uses of option cards for a system run are explained under "System Run."

The page number, line number, and program identification portions of the Autocoder entry are ignored for all option cards; therefore, this information need not be punched unless the user desires it.

1. If the source program is to be read from an input unit that differs from the one specified as a "permanent" option, use one of the following option cards:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
0 1	SOURCEDECK	USE	TAPE
0 2	SOURCEDECK	USE	S
0 3			

The first line shows the option card to be used to read the source program from a tape mounted on tape unit 2 connected to tape channel 1; however, when this option card is used *and* Alteration Switch 4 is ON to indicate that several source programs are on one tape (see "Stacking Source Programs"), tape unit 4 connected to tape channel 1 must be used for reading the tape.

The second line shows the option card to be used to read the source program from a card reader. The value of s in the operand may be 1, 2 or 3, specifying a unit record synchronizer to which an IBM 7500 Card Reader is connected.

2. To produce only an edited listing tape when on-line listing has been made a "permanent" option or to produce an on-line listing for the current run, one of the following option cards may be used:

Line	Label	Operation	OPERAND
01	REPORT LIST	USE	TAPE ONLY
02	REPORT LIST	USE	S
03			

The first line shows the option card to be used to eliminate the on-line listing called for by a "permanent" option established by the user. Instead of producing both an on-line listing and an edited listing tape, the Four-Tape Autocoder processor will produce only the tape for listing; the tape unit used for writing this tape will be indicated by a typewritten message.

The second line shows the option card which will cause the processor to produce an on-line listing in addition to the edited listing tape which is always produced. The value of s in the operand may be 1, 2 or 3 to specify the unit record synchronizer to which an IBM 7400 Printer is connected.

3. To produce only a tape containing the object program when on-line punching has been made a "permanent" option or to produce on-line punching for the current run, one of the following option cards may be used:

Line	Label	Operation	OPERAND
01	OBJECT DECK	USE	TAPE ONLY
02	OBJECT DECK	USE	S
03			

The first line shows the option card to be used to eliminate on-line punching of the object program when punching is called for by a "permanent" option established by the user. When this card is used, the Four-Tape Autocoder processor will produce only the tape containing the object program rather than both the tape and a deck of cards punched on-line. The tape unit used for writing this tape will be indicated by a typewritten message.

The second line shows the option card which will cause the object program to be punched into cards by an on-line card punch; this punching occurs in addition to writing of the object program on tape. The value of s in the operand may be 1, 2 or 3 to specify the unit record synchronizer to which an IBM 7550 Card Punch is connected.

4. If option cards and/or control cards are to be read from an input unit that differs from the one specified as a "permanent" option, use one of the following option cards:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
0 1	OPTIONUNIT	USE	TAPE
0 2	OPTIONUNIT	USE	S
0 3			

The first line shows the option card to be used to read control cards and other option cards from a tape mounted on tape unit 2 connected to tape channel 1; whenever option cards and/or control cards are to be read from tape, the tape must be read from this unit.

The second line shows the option card to be used to read option cards and/or control cards from a card reader. The value of s in the operand may be 1, 2 or 3, specifying a unit record synchronizer to which an IBM 7500 Card Reader is connected.

The OPTIONUNIT option card must always be read from the input unit established by a "permanent" option.

5. Error messages produced by Phases 1 through 4 of the processor will always be printed at the end of the program listing. The messages may also be produced separately on a typewriter or an on-line printer by using one of the following option cards:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
0 1	MESSAGES	USE	TYPEWRITER
0 2	MESSAGES	USE	S
0 3	MESSAGES	USE	TAPEONLY
0 4			

The first line shows the option card to be used to have the messages typed on the console typewriter in addition to being printed following the program listing.

The second line shows the option card to be used to have the messages printed on an on-line printer in addition to appearing at the end of the program listing. The value of s in the operand may be 1, 2 or 3 to specify the unit record synchronizer to which an IBM 7400 Printer is connected; when this option is used, the 7400 Utility Panel must be used with the printer alteration switches set to BBAA or BBBA.

The third line shows the option card to be used when the messages are to be printed only following the program listing. (This option is assumed on the Systems Tape obtained from IBM.)

Messages produced by System Control and messages accompanying programmed halts will always be typed regardless of the option used for the Phase 1 through 4 messages.

6. If the number of storage locations available for running the object program differs from the number specified by the "permanent" option, an option card punched as follows must be used:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
0 1	MEMORYSIZE	OBJCT	NNNN
0 2			

The value of NNNN in the operand specifies the storage location one greater than the highest location that may be used by the object program at the time it is executed; NNNN must be less than, or equal to, 9990.

7. If the number of storage locations available to Four-Tape Autocoder for assembly of the program is different from the number specified by the "permanent" option, an option card punched as follows must be used:

Line	Label	Operation	OPERAND
0 1	MEMORYSIZE	EQUIP	NNNN
0 2			

The value of NNNN in the operand may be any number from 5000 through 9990 which specifies the storage location one greater than the highest location that may be used by the Four-Tape Autocoder processor during the assembly of a program.

8. If input to the processor is on tape and the presence or absence of labels is not as specified by the "permanent" option, one of the following option cards may be used:

Line	Label	Operation	OPERAND
0 1	LABELSIN	EQUIP	YES
0 2	LABELSIN	EQUIP	NO
0 3			

The first line shows the option card to be used to indicate that the input tape has a standard header label written at the density specified by the TAPDENSITY EQUIP option. The format of header labels is described in the 7070 Data Processing System Bulletin "IBM 7070 Input/Output Control System," form J28-6033-1.

The second line shows the option card which indicates that the input tape does not have any labels.

Input tapes may or may not have standard trailer labels; trailer labels are ignored regardless of the option card used or the "permanent" option on the System Tape.

9. If the absence or presence of standard tape labels on work tapes used by the processor is not as specified by the "permanent" option, one of the following may be used:

Line	Label	Operation	OPERAND
0 1	LABELSOUT	EQUIP	YES
0 2	LABELSOUT	EQUIP	NO
0 3			

The first line shows the option card which indicates that the work tapes to be used by the processor have standard header labels written at the density specified by the TAPDENSITY EQUIP option. The Four-Tape Autocoder processor will change the File Identification and Creation Date fields of the header label; all other fields will remain unchanged.

The second line shows the option card to be used to indicate that the work tapes do not have standard header labels. The Four-Tape Autocoder processor will create header labels and write them on the work tapes at the density specified by the TAPDENSITY EQUIP option.

Regardless of the option card used or the "permanent" option on the System Tape, the tapes containing the edited listing and the object program will always have tape labels except when the output of several runs are "stacked" on one tape as explained below. However, a System Tape produced during a system run will never have labels.

10. If the density of the input and output tapes is to be other than that specified by the "permanent" option, one of the following cards must be used:

Line	Label	Operation	OPERAND
3 5	6 15	16 20	21 25 30 35 40 45
0 1	TAPDENSITY	EQUIP	HIGH
0 2	TAPDENSITY	EQUIP	LOW
0 3			

The first line shows the option card which specifies that the input tape, the object program tape and the program listing tape will have records and labels, if any, written in high density (556 characters per inch). If the LABELSOUT EQUIP YES option is used, the labels on the work tapes must be written in high density. The second line shows the option card which specifies that the input tape, the object program tape and the program listing tape will have records and labels, if any, written in low density (200 characters per inch). If the LABELSOUT EQUIP YES option is used, the labels on the work tapes must be written in low density.

If the TAPDENSITY EQUIP option is to be used for *temporarily* changing the tape density, the "permanent" option unit must be a card reader. If the "permanent" option unit is to be changed temporarily to a tape unit, the OPTIONUNIT USE TAPE card must *follow* the TAPDENSITY EQUIP card in the card reader because the TAPDENSITY EQUIP option cannot be read from tape, i.e., tape density cannot be changed once reading from tape has begun.

Stacking Input and Output

If more than four tape units are available in the 7070 or 7074 Data Processing System, the extra tape unit(s) may be used to "stack" the input or output of several runs. "Stacking" refers to the writing of the input or output of several programs successively on one reel of tape. Stacking permits processing of many programs while reducing the number of manual operations, i.e., tape reels need not be changed after processing each source program. Depending on the number of tape units available in addition to the initial four and the tape channels to which they are connected, the user may stack any or all of the following:

1. Source programs if an extra tape unit is on channel 1.
2. Edited program listings if an extra tape unit is on channel 1.
3. Object programs if an extra tape unit is on channel 2.

The options of stacking the input and output are selected through the console alteration switches.

Stacking Source Programs

Several source programs for successive original assembly runs may be read from a tape mounted on tape unit 4 connected to channel 1 whenever Alteration Switch 4

is ON and the SOURCEDECK USE TAPE option is used. The tape containing the source programs must be written so that each source program is followed by a tape mark.

Each program is independent of the others on the tape; therefore, each source program must be preceded by an ORIGINAL RUN Control Card and any option cards that may be desired. (The option cards may be read from an on-line card reader instead of including them on the tape.)

If the LABELSIN EQUIP YES option is in use, the first card of each source program deck written on tape must be a "dummy" card which acts as a tape label; a blank card may be used as the "dummy" card. Alternately, if option cards are being read from an on-line card reader, the label option may be changed temporarily by using a LABELSIN EQUIP NO option card. Trailer labels must not be used regardless of the LABELSIN EQUIP option in effect.

The tape containing the stacked source programs will never be rewound by Four-Tape Autocoder.

Tape unit 2 connected to channel 1, normally used for individual (not stacked) source program input, is used as a work tape when source programs are stacked.

Stacking Edited Listings

The edited listings of several programs processed by Four-Tape Autocoder may be written on tape unit 5 connected to channel 1 whenever Alteration Switch 2 is ON. The listing of each object program will be followed by a tape mark to identify the end of the listing.

The tape containing the stacked listings will not have tape labels, will never be rewound by Four-Tape Autocoder, and may not be used as the input to a re-assembly run.

Stacking Object Programs

The object programs in Condensed Card Format, produced by several Four-Tape Autocoder runs, may be written on tape unit 2 connected to channel 2 whenever Alteration Switch 3 is ON. The condensed card deck of each object program will be followed by a tape mark to identify the end of the program.

The tape containing the stacked object programs will not have tape labels and it will never be rewound by Four-Tape Autocoder. If only channel 1 is to be used for processing (see "Operating Instructions"), object programs may not be stacked.

Operating Instructions

Each of the three types of runs possible with Four-Tape Autocoder requires slightly different preparations before a run can be started. Once the 7070 or 7074 system has been set up for a particular type of run, the same console procedure may be used to start the run. Preparations required for each type of run are described separately below and are followed by one group of instructions pertaining to console operations.

The System Tape that is available from the IBM 7070/7074 Program Librarian assumes a typical 7070 or 7074 Data Processing System that might be used for assembling programs with Four-Tape Autocoder. This typical system is described by the following "permanent" options included as part of the System Tape:

1. Option cards, control cards and the source program will be read from tape.
2. No on-line unit record machines will be used.
3. The 7070 or 7074 Data Processing Systems to be used by the object program and the processor both have 9,990 words of core storage available.
4. Neither the source program tape nor any of the work tapes will have labels.

5. The source program tape, the program listing tape and the object program tape will be in low density, i.e., 200 characters per inch.

6. Error messages will be printed only at the end of the program listing.

If these "permanent" options describe the user's 7070 or 7074 system, no option cards need be included when assemhling with Four-Tape Autocoder. However, if the user's system differs from the assumed one or if additional options are desired, the "permanent" options on the System Tape may he changed or new options added through a system run; new options may also be added or substituted for a particular run by the use of option cards ahead of the RUN Control Card.

To simplify the description of the preparations for each type of run, option cards and control cards which may be read from either cards or tape will be explained in terms of "cards" although they may actually he on tape. For example, if the "permanent" option unit is a tape unit, the statement "Place option cards in the permanent option unit," means that the option cards must be written on a tape which is then mounted on the "permanent" option unit.

The level number of the Four-Tape Autocoder System Tape will he typed at the head of the console typewriter output for each run. The level number will also be printed on the first heading line of each page of the edited program listing.

In the operating instructions given below, tape units are referred to by a combined channel and unit number. For example, tape unit 21 refers to tape unit 1 connected to channel 2.

Preparations for an Original Assembly

1. Mount the Four-Tape Autocoder System Tape on tape unit 20. If only one channel is available, mount the System Tape on tape unit 10. Set the tape unit to high density.

2. Mount two work tapes as follows:

	Tape Channel	Tape Unit
2 Channel Operation	1	3
	2	1
1 Channel Operation	1	3
	1	1

3. Place the option cards, if any, in the "permanent" option unit. If an OPTIONUNIT option card is included, any option card(s) which follows the OPTIONUNIT card must be placed in the unit specified by the OPTIONUNIT card.

4. Place an ORIGINAL RUN Control Card in the "permanent" option unit; if an OPTIONUNIT option card is included in step 3, the RUN Control Card must be placed in the unit specified by the OPTIONUNIT card.

5. Prepare the source program to be assembled as follows:

   A. If the source program has been written on tape by a card-to-tape operation, mount the tape on tape unit 12. Set the tape unit to the density specified by the TAPDENSITY EQUIP option. (The source program may occupy one full reel of tape.)

   B. If the source program is in cards, place the source program deck in the on-line card reader, insert the 7500 Utility Panel and set the card reader

alteration switches to BBAA. When an on-line card reader is used for the source program, mount a work tape on tape unit 12.

6. If a "permanent" option on the System Tape or an option card included in step 3 specifies on-line printing or punching, insert the 7400 Utility Panel or the 7550 Utility Panel into the on-line printer and punch, respectively. Set the printer alteration switches to AAAA; set the punch alteration switches to AAAA to have the load card indicator (12 punch) punched into column 65 or ABAA to have it punched into column 79.
7. Continue as explained under "Console Procedure."

Preparations for a Re-assembly

1. Mount the Four-Tape Autocoder System Tape on tape unit 20. If only one channel is available, mount the System Tape on tape unit 10. Set the tape unit to high density.
2. Mount a work tape on tape unit 21. If only one channel is available, mount the work tape on tape unit 11.
3. Mount the edited listing tape produced by an original assembly or a previous re-assembly on tape unit 13.
4. Place the option cards, if any, in the "permanent" option unit. If an OPTIONUNIT option card is included, any option card(s) which follows the OPTIONUNIT card must be placed in the unit specified by the OPTIONUNIT card.
5. Place a RECOMPILE RUN Control Card in the "permanent" option unit; if an OPTIONUNIT option card is included in step 4, the RUN Control Card must be placed in the unit specified by the OPTIONUNIT card.
6. Arrange the corrections to the source program in page number and line number sequence (see "Re-assembly Run" for card format used for corrections). Prepare the corrections for the re-assembly run as follows:
   A. If the corrections have been written on tape by a card-to-tape operation, mount the tape on tape unit 12. Set the tape unit to the density specified by the TAPDENSITY EQUIP option. (The corrections may occupy one full reel of tape.)
   B. If the corrections are in cards, place the cards in the on-line card reader, insert the 7500 Utility Panel and set the card reader alteration switches to BBAA. When an on-line card reader is used for reading the corrections, mount a work tape on tape unit 12.

   Note: At least one correction card must be supplied for each re-assembly run.
7. If a "permanent" option on the System Tape or an option card included in step 4 specifies on-line printing or punching, insert the 7400 Utility Panel or the 7550 Utility Panel into the on-line printer and punch, respectively. Set the printer alteration switches to AAAA; set the punch alteration switches to AAAA to have the load card indicator (12 punch) punched into column 65 or to ABAA to have it punched into column 79.
8. Continue as explained under "Console Procedure."

Preparations for a System Run

1. Mount the Four-Tape Autocoder System Tape on tape unit 20. If only one channel is available, mount the System Tape on tape unit 10. Set the tape unit to high density.

2. Mount the two work tapes as follows:

	Tape Channel	Tape Unit
2 Channel Operation	1	3
	2	1
1 Channel Operation	1	3
	1	1

3. Place the option cards, if any, in the "permanent" option unit. If an OPTIONUNIT option card is included, any option card(s) which follows the OPTIONUNIT card must be placed in the input unit specified by the OPTIONUNIT card.
4. Place a SYSTEMS RUN Control Card in the "permanent" option unit; if an OPTIONUNIT option card is included in step 3, the RUN Control Card must he placed in the unit specified by the OPTIONUNIT card.
5. Group the changes to the System Tape as follows (if no changes are used, a duplicate of the System Tape will he produced):

   Place changes to "permanent" options behind one ASIGN card.

   Place all additions and/or changes to any item(s) in the Library behind an INSER card for that item(s). (DELET cards need not be grouped.)

   Place changes to the Four-Tape Autocoder processor hehind an UPDAT card(s) which identifies the section(s) of the processor that is to be changed.

   The various groups of changes may be in any order, e.g., the ASIGN card followed hy option changes may follow changes to the Library. Card formats for the various changes to the System Tape are described under "System Run." Prepare the changes for the system run as follows:

   A. If the changcs have been written on tape by a card-to-tape operation, mount the tape on tape unit 12. If the density has been specified by a TAPDENSITY EQUIP option card, set the tape unit to that density; if the original density option (the one on the System Tape supplied by IBM) has not been changed, set the tape unit to low density.

   B. If thc changcs are in cards, place the cards in the on-line reader, insert the 7500 Utility Panel and set the card reader alteration switches to ABAA; cards which are not in the load card format may not contain a plus sign (12 punch) in column 79. When an on-line card reader is used for reading the changes, mount a work tape on tape unit 12.

6. Continue as explained under "Console Procedure."

Console Procedure

1. Manually store the following three instructions where c is either a 1 or a 2 to indicate whether 1 or 2 channels are to hc used for the run:

Storage Location	Instruction to be Stored
0000	−8C01010004
0001	−0100030010
0002	+5100C10002

2. Set 7070 console switches as follows:

   Turn Alteration Switch 1 ON to cause certain halts. (See Halts 1111, 2222, 0031 and 3333.)

   Turn Alteration Switch 2 ON if edited program listings are to be stacked on tape unit 15.

   Turn Alteration Switch 3 ON if object programs are to be stacked on tape unit 22.

   Turn Alteration Switch 4 ON if source programs are stacked on tape unit 14 and mount a work tape on tape unit 12.

   Turn all accumulator overflow switches OFF.

   Turn both Unit Record Priority Controls to OFF and N.

3. Depress the Run key.
4. Depress the Computer Reset key.
5. Depress the Start key.

Halts and Messages

When a halt occurs in program operation, the contents of the instruction counter and the program register will be automatically typed on the console typewriter. The halts and associated messages which may occur during a Four-Tape Autocoder assembly are listed below, together with the causes and appropriate action. The programmed halts are arranged in numerical order according to the halt number, i.e., the last four digits of the contents of the program register typed when the halt occurs.

Programmed halts executed by the IBM 7070 Input/Output Control System during Four-Tape Autocoder processing are listed separately following the processor halts.

Halt Number	Explanation and Action
0000	END OF SYSTEM RUN, NEW SYSTEM TAPES ON C1 AND 12. A system run has been completed. A new System Tape is on each of the tape units named in the message. The value of *C* will be either a 1 or a 2 depending on the number of tape channels used.
0001	CARD READ ERROR A card reader validity error has occurred while reading either an option card or a library change card; the card read when the error occurred is the last card in the stacker. Replace the last card, and all cards following it, in the hopper. Depress the Start key on the card reader and the Start key on the console.
0002	END OF FILE WHILE READING OPTION CONTROL CARDS The last option card was not followed by a RUN Control Card. Insert the appropriate RUN Control Card and start the assembly from the beginning.

	Halt Number	Explanation and Action
Phase 1		
	0011	*PGLIN* CARD READ ERROR A card reader validity error has occurred while reading the source program. The page number and line number of the card read when the error occurred appears in the message in place of *PGLIN;* this card is the last card in the stacker. Replace the last card, and all cards following it, in the hopper. Depress the Start key on the card reader and the Start key on the console.
	1111	END PHASE 1 This message is always typed but the machine will halt only if Alteration Switch 1 is ON at the end of Phase 1; Phase 2 will be entered automatically if Alteration Switch 1 is OFF. This halt is provided for users who wish to save tapes used as input to Phase 1. When the halt occurs, replace the source input tapes (channel 1, unit 2) and, for a re-assembly run, the previous edited listing tape (channel 1, unit 3) with work tapes. After the work tapes have been mounted, depress the Start key to begin Phase 2.
Phase 2		
	2222	END PHASE 2 This message is always typed but the machine will halt only if Alteration Switch 1 is ON at the end of Phase 2; Phase 3 will be entered automatically if Alteration Switch 1 is OFF. Depress the Start key to begin Phase 4 after this halt.
Phase 3		
	0031	SYMBOL TAPE *CU* PROCESSED This programmed halt and message occurs at the end of the symbol processing portion of the first pass of Phase 3 only if Alteration Switch 1 is ON. If the user desires to preserve the symbol tape, it may be removed from the tape unit indicated by channel and unit number (*CU*) in the message and replaced by a work tape. Depress the Start key to resume Phase 3.
	3333	END PHASE 3 This message is always typed but the machine will halt only if Alteration Switch 1 is ON at the end of Phase 3; Phase 4 will be entered automatically if Alteration Switch 1 is OFF. Depress the Start key to begin Phase 4 after this halt.
Phase 4		
	0042	*(No Message)* An error has occurred during a Lookup Lowest operation. Depress the Start key to repeat the operation.
	4444	ASSEMBLY COMPLETED LISTING ON TAPE *XX* PROGRAM ON TAPE *YY* An original assembly or a re-assembly run has been completed. The edited listing tape is on tape unit *XX* and the assembled program in condensed card format is on tape unit *YY*. The symbols *XX* and *YY* each represent the channel and unit numbers of the tape units holding the corresponding output tapes.

Halt Number	Explanation and Action
2910	TPZZ COUNT DTF-*XXXXX* TRL-*YYYYY* The block count, *XXXXX*, (stored in the DTF) made while reading tape ZZ does not agree with the block count in the trailer label on input tape ZZ. The operator may ignore the discrepancy by depressing the Start key.
2911	One, two or three messages will be typed when this halt occurs. TPZZ RN ERR LI-*XXXXX* HDR-*YYYYY* The reel sequence number, *XXXXX*, in the label information DC for input tape ZZ does not agree with the reel sequence number, *YYYYY*, in the header label on tape ZZ. TPZZ FS ERR LI-*XXXXX* HDR-*YYYYY* The file serial number, *XXXXX*, in the label information DC for input tape ZZ does not agree with the file serial number, *YYYYY*, in the header label on tape ZZ. TPZZ ID ERR LI-*XXXXX* HDR-*YYYYY* The file identification, *XXXXX*, in the label information DC for input tape ZZ does not agree with the file identification, *YYYYY*, in the header label on tape ZZ. The five *X*s and *Y*s represent either the first five characters of the file identification, if they do not agree, or the last five characters of the file identifications, if the first five characters of both agree. When this halt occurs, the error(s) may be corrected by mounting the proper reel of tape, depressing the Program Reset key and then depressing the Start key. If desired, the discrepancy may be ignored by depressing the Start key.
2912	One of two forms of messages will be typed when this halt occurs. TP*XX* LAB ERR C*Y* An unusual condition, identified by condition code *Y*, occurred when reading a label on tape *XX*. Probably there is no label on the tape. Either depress the Start key to reread the label or change to another reel of tape and then depress the Start key. TP*XX* TM ERR C*Y* An unusual condition, identified by condition code *Y*, occurred when writing a tape mark on tape *XX*. To try to write the tape mark again, depress the Start key.
2960	*(No Message)* An error in writing a segment mark or tape mark has occurred. Depress the Start key to try writing five more times.
2961	TP*XX* RDSM ERR*Y* An unusual condition, identified by condition eode *Y*, occurred when reading a segment mark on tape *XX*. To ignore the error, depress the Start key.

Messages

Messages which may be typed on the console typewriter during processing by each phase of Four-Tape Autocoder are listed below in approximate alphabetical order for ease of reference. When a message indicates an error condition, the operator has the option of discontinuing the run immediately or allowing the run to continue to the end. It is generally advisable to finish an original assembly or a re-assembly run so that all errors will have been noted; errors may be corrected manually by changing the condensed card output (object program) or automatically by preparing corrections for a subsequent re-assembly. Of course, if the errors are too numerous, the source program should be corrected and a new assembly started. When errors in a system run occur, it is advisable to correct the source material and start the system run from the beginning.

In addition to the typed list of error messages, the listing of the assembled program will usually indicate the instruction which caused an error message to be typed. Each message, except those preceded by an asterisk in this manual, will correspond to a line in the program listing that has the word ERROR printed to the right of the assembled instruction. The symbol *PGLIN* used as part of messages in this list will be replaced by the page number (*PG*) and line number (*LIN*) of the entry which is in error.

Messages typed by the IBM 7070 Input/Output Control System during Four-Tape Autocoder processing are listed separately following the processor messages.

System Control and Librarian

*DENSITY OPTION NOT PROCESSED

A change in density has been specified after part of the input tape has been read; the specified change will be ignored. Four-Tape Autocoder will process the source program in the density specified by the "permanent" option.

**PGLIN* INVALID OPTION CONTROL CARD

An error in preparing an option card has occurred; the option card will be ignored. If the intended option cannot be ignored, correct the option card and restart the program from the beginning.

**PGLIN* INVALID LIBRARY CONTROL CARD

An error in preparing a library control card for a system run has occurred; the control card will be ignored. If the library control card cannot be ignored, correct the control card and restart the system run from the beginning.

*MACRO TABLE EXCEEDS MAXIMUM SIZE

The macro-instructions and/or subroutines to be added to the Library during this system run would exceed the maximum number (196) of items that the Library may contain. Remove macro-instructions and/or subroutines (or use DELET cards to delete items already in the Library) so that the total number of items is 196 or less; restart the system run from the beginning.

**MNAME* COUNT DOES NOT AGREE
**SNAME* COUNT DOES NOT AGREE

The number of cards specified in the operand of an INSER card does not agree with the number of cards actually used for the macro-instruction (*MNAME*) or subroutine (*SNAME*). If this error cannot be ignored, correct the INSER card and/or the macro-instruction or subroutine; restart the system run from the beginning.

PGLIN MNAME NOT FOUND IN MACRO TABLE
PGLIN SNAME NOT FOUND IN MACRO TABLE

The macro-instruction (*MNAME*) or subroutine (*SNAME*) used in a DELET card is not in the macro table; the DELET card will be ignored. If the intended deletion cannot be ignored for this system run, correct the DELET card and restart the system run from the beginning.

*INCORRECT CORRECTION CARD FORMAT PHASE *X*

A card containing corrections to a section of the Four-Tape Autocoder processor is not in condensed card format. The value of *X* will be a digit from 0 through 4 to indicate the section of the processor for which the correction was intended. The digit 0 indicates the System Control and Librarian section; digits 1 through 4 indicate Phases 1 through 4 respectively.

The card containing the change will be ignored and the system run will continue. If the change to the processor cannot be ignored, correct the error and restart the system run from the beginning.

Phase 1

*PROGRAM NUMBER *XXXXX*

This message is typed for each assembly or re-assembly to identify the program being processed. The program identification (*XXXXX*) will be the same as the contents of columns 76 through 80 of the first card of the source program.

PGLIN ADDRESS ADJUSTMENT ERROR

A parameter in either the operand of a macro-instruction or the operand of a skeleton instruction has two address adjustment figures, e.g., FIELD+16+8. The first address adjustment figure will be ignored and the operand processed as though only the second figure were present.

*$PGLIN_1$ AFTER $PGLIN_2$

The card identified by $PGLIN_1$ is out of page and line number sequence; it follows the card identified by $PGLIN_2$. This out-of-sequence condition will be ignored and the card ($PGLIN_1$) processed as though it should follow the $PGLIN_2$ card.

PGLIN XXXXX ASSUMED BLANK

The Operation columns of a subsequent entry under a DTF descriptive entry (DTF entries are part of the 7070 Input/Output Control System) are not blank but contain the characters (*XXXXX*) printed in the message. This message indicates that either the Operation columns of the entry were punched by mistake or an insufficient number of subsequent entries for the DTF are present. The Operation columns will be processed as blanks, i.e., as a legitimate entry under the DTF.

If it is known that this message was caused by an insufficient number of entries under the DTF rather than by a card punching error, it may be desirable to stop the assembly if the Operation Code (*XXXXX*) assumed to be blank will introduce many errors into other portions of the program.

PGLIN FIELD DEFINITION ERROR

A parameter in either the operand of a macro-instruction or the operand of a skeleton instruction has incorrect field definition.

**PGLIN* IMPROPER DELET

An improperly punched DELET card has been detected during a re-assembly run. If the *PGLIN* does not correspond to a *PGLIN* of an Autocoder entry in

the program being re-assembled, the DELET card will be ignored and the deletions indicated by this card will not occur. If the *PGLIN* corresponds to an entry but the operand does not, as many consecutive Autocoder entries as do exist in the sequence will be deleted from the program being re-assembled.

PGLIN INDEXING ERROR

Indexing of a parameter has been specified in the operand of a macro-instruction for a paramcter that already has indexing specified in the skeleton instruction. The indexing in the skeleton instruction will be used; the indexing in the operand of the macro-instruction will be ignored.

**PGLIN* MACRO SELECTION NOT POSSIBLE

The operand of a macro-instruction provided by the IBM 7070 Input/Output Control System is incorrect. The macro-instruction will be omitted from the assembled program; it will appear in the edited listing with no corresponding machine-language instruction(s).

PGLIN OPERAND ASSUMED 0

The operand of a subsequent entry under an Input/Output Control System DTF descriptive entry is incorrect; the operand will be assumed to contain a 0 and processed accordingly.

PGLIN OPERAND ASSUMED 1

The operand of a subsequent entry under an Input/Output Control System DTF descriptive entry is not within its allowable range; the operand will be assumed to contain a 1 and processed accordingly.

PGLIN OPERAND ERROR

The operand of the header line of a DTF descriptive entry does not begin with an alphabetic character; Four-Tape Autocoder will process the DTF entry as though the operand were blank. This condition may introduce errors in entries which are intended to refer to this file. If this file is referred to frequently in the source program, it is recommended that the DTF entry be corrected and the assembly restarted from the beginning.

PGLIN PARAMETER MISSING

A parameter which must always be present in the operand of a macro-instruction (specified by ¤1 through ¤9 in the skeleton routine) has been omitted. Four-Tape Autocoder will attempt to complete the macro-instruction; this will probably cause other error messages to be typed in the following phases of the assembly.

PGLIN SNAME NOT IN LIBRARY

A subroutine or the skeleton routine of a macro-instruction uses an INCL operation which names a subroutine that is not in the Library. The INCL entry will be ignored and the subroutine or macro-instruction assembled without it.

Phase 2

PGLIN XXXXX ASSUMED NOP

The Operation columns of an entry contain an invalid Operation Code (*XXXXX*); the entry will be assembled as though the Operation Code were NOP.

PGLIN FIELD DEFINITION ERROR

The operand of an Autocoder entry has incorrect field definition.

PGLIN LABEL ERROR

The symbolic label of an Autocoder entry has an incorrect format. Four-Tape Autocoder will ignore the label and process the entry as though the Label columns were blank.

PGLIN XXXXXXXXXX NOT PREVIOUSLY DEFINED

The symbol (*XXXXXXXXXX*) has not appeared as a label earlier in the program sequence although it is used in the operand of an Autocoder entry which requires that the symbol be defined in this manner.

**PGLIN* NUMBER OF WORDS ASSUMED 1

This message will be typed if there are no subsequent entries following a DA or DC header line or if the number of words reserved by a DA or DC entry exceeds 999. For either error, the DA or DC entry will be processed as though one word were indicated.

PGLIN OPERAND OUT OF RANGE

The actual number used in the operand of an Autocoder entry to specify channel, channel and unit, arm and file, stacking latch, unit record synchronizer, index word or electronic switch is not within the range of valid numbers for the item being specified. For example, if the number 92 were used to specify tape channel and unit, it would be out of the range of valid numbers for tape channel and unit, i.e., 10 through 49.

PGLIN SHIFT AMOUNT ERROR

The number of positions to be shifted using an uncoupled shift exceeds 10 or the number for a coupled shift exceeds 20. The entry will be processed as though the shift number were 0.

PGLIN STORAGE LIMIT EXCEEDED

The number of storage locations available for the object program has been exceeded. This error may be caused by an error in the MEMORYSIZE OBJCT option card used for the assembly. Four-Tape Autocoder will continue assigning locations even though the location falls above the limit indicated for the object machine. Assignment of items above the storage limit will cause additional error messages in later phases of the processor when these locations are used; therefore, if many items are assigned above the limit, it is advisable to restart from the beginning using the correct MEMORYSIZE OBJCT card.

Phase 3

PGLIN XXXXXXXXXX ADDRESS OUT OF RANGE

The symbol *XXXXXXXXXX* in the operand of an Autocoder entry has been assigned a storage location above the limit specified for the object program. This error may be caused by an error in the MEMORYSIZE OBJCT option card; see last message for Phase 2. Four-Tape Autocoder will process this entry as though the assigned location were within the range of the object machine.

XXXXXXXXXX ASSIGNED +0000*FFNNNN*
XXXXXXXXXX ASSIGNED +0000*FFNNNN*

These messages will always appear in pairs to indicate that a symbol (*XXXXXXXXXX*) has been used as a label of two source program entries that either specify different field definition and/or have been assigned to different locations. The message will show the field definition (*FF*) and the actual storage location (*NNNN*) of the symbol. (These messages are not typed if the symbol is used as a label more than once but has the same field definition and storage location each time.)

PGLIN XXXXXXXXXX FIELD DEFINITION ERROR
The field definition used with the symbol *XXXXXXXXXX* to define a subfield produces an error in relative field definition digits.

PGLIN XXXXXXXXXX NO MORE INDEX WORDS AVAILABLE
A 7070 index word cannot be assigned to symbolic index word *XXXXXXXXXX* because index words 1 through 96 have already been assigned. No assignment will occur and instructions using unassigned symbolic index words will contain zeros in place of an index word number.

This message will be typed only for the first unassignable symbolic index word; additional unassignable index words will be processed as explained above but no message will be typed.

PGLIN XXXXXXXXXX NO MORE SWITCHES AVAILABLE
A 7070 electronic switch cannot be assigned to symbolic electronic switch *XXXXXXXXXX* because electronic switches 1 through 30 have already been assigned. No assignment will occur and instructions using unassigned symbolic electronic switches will contain zeros in place of an electronic switch number.

This message will be typed only for the first unassignable symbolic electronic switch; additional unassignable electronic switches will be processed as explained above but no message will be typed.

PGLIN XXXXXXXXXX NOT A SYMBOLIC INDEX WORD
The symbol *XXXXXXXXXX* is used as a symbolic index word in the Autocoder entry but it has been defined previously as something other than an index word.

PGLIN XXXXXXXXXX NOT A SYMBOLIC SWITCH
The symbol *XXXXXXXXXX* is used as a symbolic electronic switch in the Autocoder entry but it has been defined previously as something other than an electronic switch.

PGLIN XXXXXXXXXX SYMBOLIC COMPONENT OUT OF RANGE
The source program has caused the processor to assign an invalid actual number to the symbolic name (*XXXXXXXXXX*) of a channel, channel and unit, arm and file, stacking latch or unit record synchronizer; the actual number is not within the range of valid numbers for the item being specified by the symbol. The instruction will be assembled using the units or tens and units positions of the actual number assigned.

Phase 4

PGLIN XXXXXXXXXX FIELD DEFINITION ERROR
The field definition used with the symbol *XXXXXXXXXX* to define a subfield produces an error in relative field definition digits.

PGLIN PRINTER ERROR
A validity error has occurred during the printing of the listing on an on-line IBM 7400 Printer. The line will be printed using the Unit Record Write Invalid (UWIV) instruction.

PGLIN XXXXXXXXXX UNDEFINED
The symbol *XXXXXXXXXX* appears in the operand of this Autocoder entry but the symbol does not appear as a label of any entry in the source program.

*PUNCH ERROR, CARD # *XXXXX*
An error in punching a card of the object program (in condensed card format) on an on-line IBM 7550 Card Punch has occurred. The number (*XXXXX*)

typed in the message is the serial number of the card (card columns 71 through 75); the card will be offset in the punch stacker.

*TOO MANY LITERALS ADDRESSES OUT OF RANGE

Some or all of the literals used by the object program have been assigned storage locations above the limit specified for the object program. This error may be caused by an error in the MEMORYSIZE OBJCT option card. Four-Tape Autocoder will process entries referring to the literals as though the assigned locations were within the range of the object machine.

Input/Output Control System

Some of these messages will be followed by a machine halt to allow the operator to make corrections, if necessary. Any halt associated with these messages is merely procedural; therefore, the halt number has no significance.

CHGE TP*XX*

Tape *XX* has been processed and the next reel of the file is to be mounted on the same tape unit. Mount the next reel on tape unit *XX* and depress the Start key.

CHK DISABLE ON

The Input/Output Control System is about to search an area for invalid words. Set the Check Disable key on the Customer Engineering Console to the DISABLE position and then depress the Start key.

CHK DISABLE OFF

The search for invalid words has been completed; all invalid words have been corrected. Set the Check Disable key on the Customer Engineering Console to the CHECK position and then depress the Start key.

ERR *NNNNYYYY*

The Input/Output Control System error routine has found an error in location *NNNN*. The contents, *YYYY*, of that location will be typed and the machine will halt to allow the operator to correct the error. After the contents of location *NNNN* have been corrected, depress the Start key.

INVALPHA *XY*0000*ZZZZ*

The contents of digit positions *X* and *Y* in the alphameric word in storage location *ZZZZ* constitute an invalid double-digit character.

TP*XX* NOT READY

Tape unit *XX* is not in ready status. Any of the following may cause this message:

1. There is no tape on the tape unit.
2. The tape is being rewound.
3. Tape is loaded on the tape unit but the Start key (on the tape unit) was not pressed to put the unit in ready status.

The operator should do whatever is necessary to place the tape unit in ready status; no action may be required if the tape is rewinding.

TP*XX* READ FAIL

An error in reading tape *XX* was not corrected in nine rereading operations. Depress the Start key to cause the tape record in error to be typed. After the record has been typed, a halt will occur to permit the operator to correct the error.

TP*XX* READ FAIL TP WD ERR

A tape word error (unusual condition code 0) has occurred in reading tape *XX*. Depress the Start key to cause the tape record in error to be typed. After the record has been typed, a halt will occur to permit the operator to correct it.

TP*XX* EOS
TP*XX* LLR
TP*XX* SCLR
TP*XX* SLR

An unusual condition has occurred in reading from tape *XX;* the unusual condition was an End-of-Segment (EOS), a Long Length Record (LLR), a Short Character Length Record (SCLR), or a Short Length Record (SLR) as indicated in the message. Depress the Start key to ignore the unusual condition.

TP*XX* WRITE FAIL

An error in writing tape *XX* was not corrected in five rewriting operations (four of the five operations were preceded by Tape Skip instructions). Depress the Start key for five more rewriting operations.

TP*YY XXXXX* EZZ N*QQ* P*WW*

This message gives the statistics regarding the reading of input tape *YY* where:

XXXXX is the block count.

ZZ is the number of times the Input/Output Control System error routine was entered.

QQ is the number of noise errors that occurred. (A noise error is caused by extraneous bits, i.e., magnetic spots, in inter-record gaps.)

WW is the number of "permanent" errors; a "permanent" error is one that is not eliminated in nine rereading attempts.

TP*YY XXXXX* EZZ SK *QQQ*

This message gives the statistics regarding the writing of output tape *YY* where:

XXXXX is the block count.

ZZ is the number of times the Input/Output Control System error routine was entered.

QQQ is the number of Tape Skip (TSK) instructions executed during the writing of the tape.

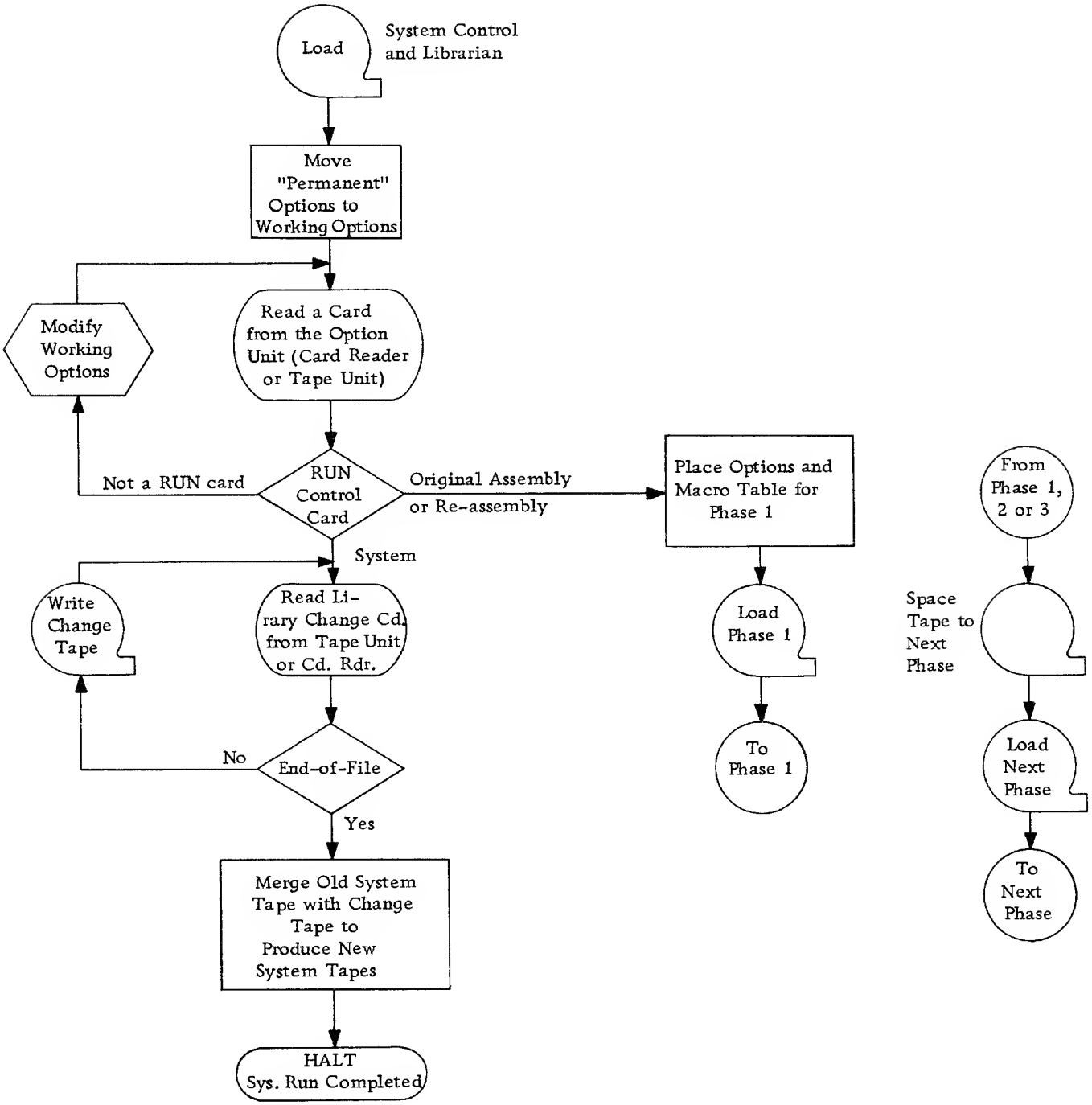
(A tape record will be typed.)

After the execution of a tape instruction, the record typed was found to be in error.

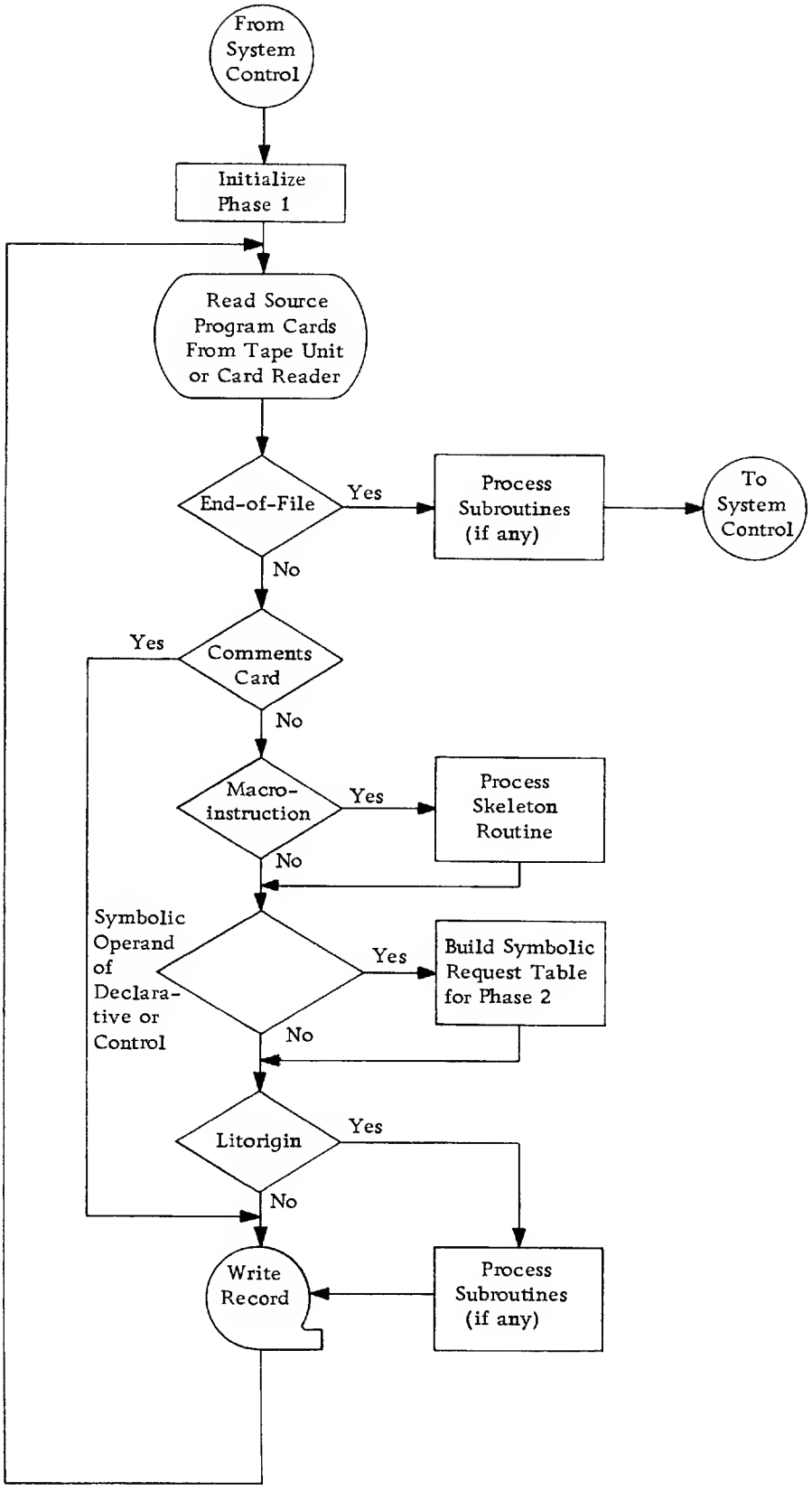
(The sixteen words of a tape label will be typed.)
A word in the tape label area does not have an alphameric sign.

Appendix A: Processor Flow Charts

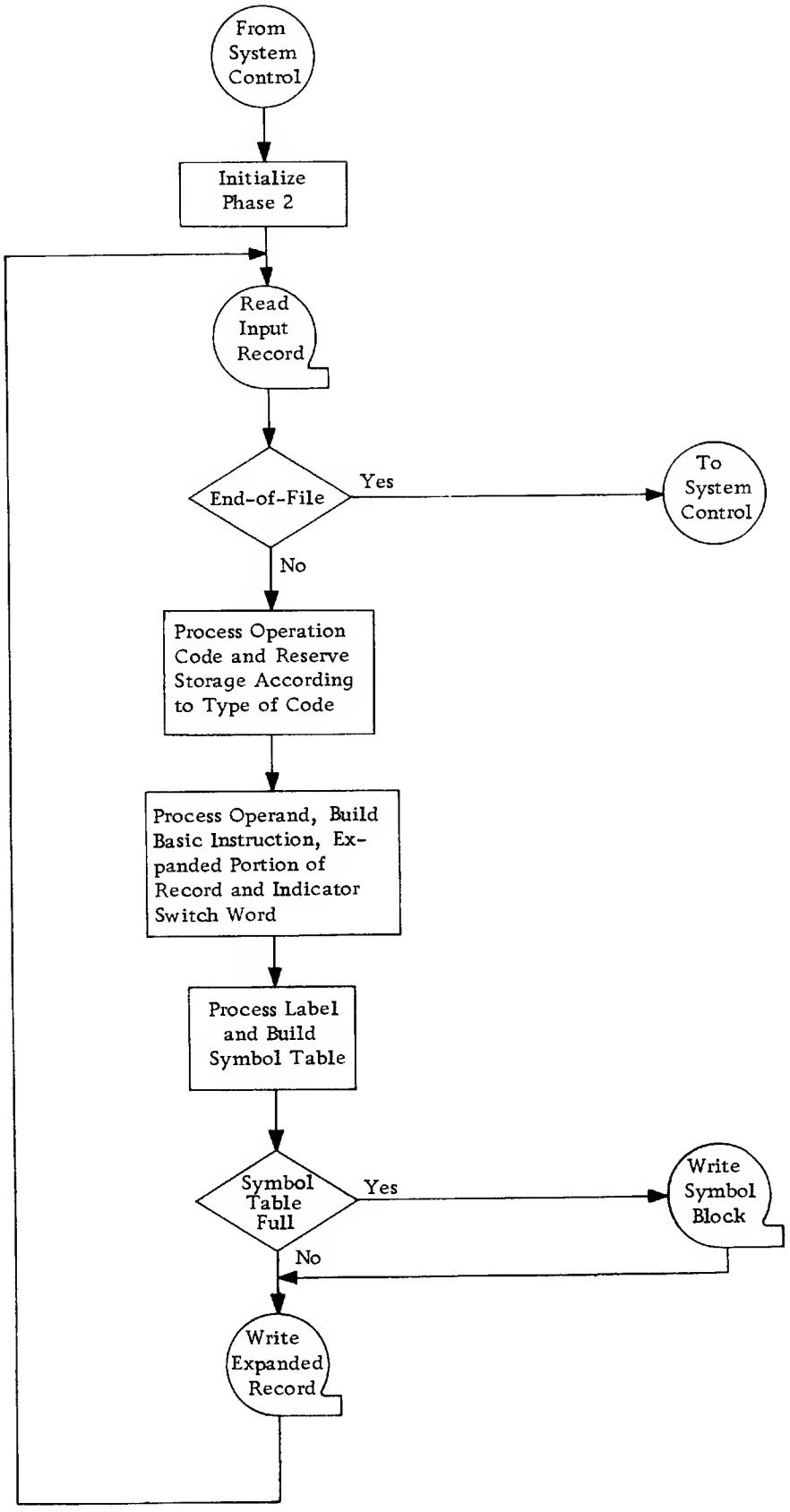
SYSTEM CONTROL AND LIBRARIAN



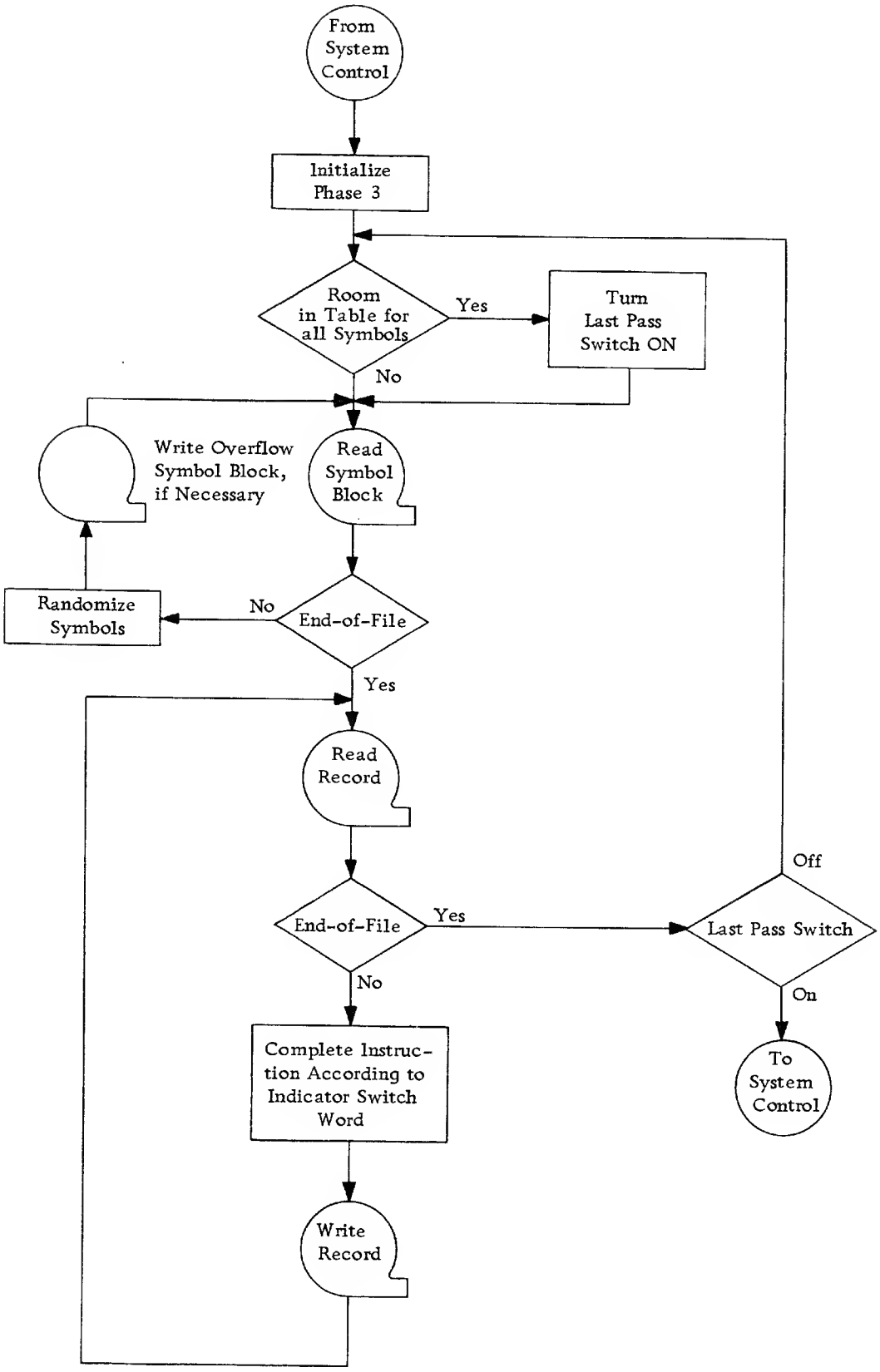
PHASE 1


From System Control
Initialize Phase 1
Read Source Program Cards From Tape Unit or Card Reader
End-of-File
Yes
Process Subroutines (if any)
To System Control
No
Comments Card
Yes
No
Macro-instruction
Yes
Process Skeleton Routine
No
Symbolic Operand of Declara-tive or Control
Yes
Build Symbolic Request Table for Phase 2
No
Litorigin
Yes
No
Write Record
Process Subroutines (if any)

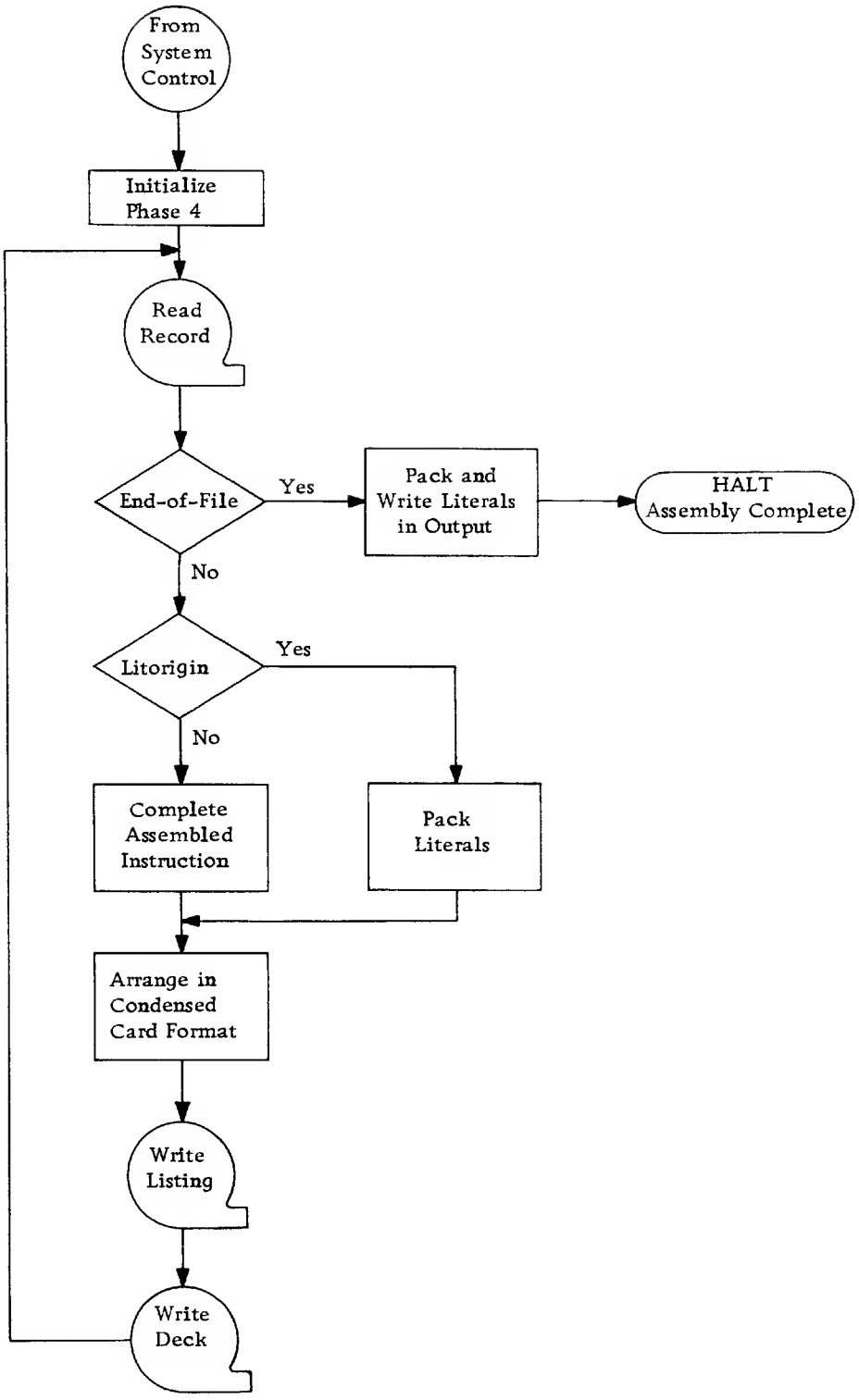
PHASE 2


From System Control
Initialize Phase 2
Read Input Record
End-of-File
Yes
To System Control
No
Process Operation Code and Reserve Storage According to Type of Code
Process Operand, Build Basic Instruction, Expanded Portion of Record and Indicator Switch Word
Process Label and Build Symbol Table
Symbol Table Full
Yes
Write Symbol Block
No
Write Expanded Record

PHASE 3


From System Control
Initialize Phase 3
Room in Table for all Symbols
Yes
Turn Last Pass Switch ON
No
Write Overflow Symbol Block, if Necessary
Read Symbol Block
Randomize Symbols
No
End-of-File
Yes
Read Record
End-of-File
Yes
Last Pass Switch
Off
On
No
Complete Instruction According to Indicator Switch Word
To System Control
Write Record

PHASE 4


From System Control
Initialize Phase 4
Read Record
End-of-File
Yes
Pack and Write Literals in Output
HALT Assembly Complete
No
Litorigin
Yes
No
Complete Assembled Instruction
Pack Literals
Arrange in Condensed Card Format
Write Listing
Write Deck

Appendix B: Alphabetic List of 7070 Autocoder Imperative Operation Codes

In the following list, the symbols in the operand column indicate what is permissible in the operand and the order in which this information must be written on the coding form.

In all cases where an "A" has been indicated, a literal may also be used. The list, however, indicates a literal, "L," and also field definition, "F," only where it would seem to be of practical value. Caution is advised when using literals with operation codes which do not specifically indicate them.

Operand Symbol Key

Symbol	Meaning	Type of Coding	Range of Actual
A	Address	Symbolic or actual	Any storage location
B	Blank		
C	Channel number	Symbolic or actual	1-4
CU	Channel and unit	Symbolic or actual	10-49
D	Digit	Actual	0-9
F	Field Definition	Actual (Enclosed in parentheses)	0-9
AF	Arm and file	Symbolic or actual	00-03, 10-13, 20-23
I	Unit record latch	A or 1, B or 2	1-2
L	Literal		
N	Number	Actual	0-10 (for normal shifts) 0-20 (for coupled or split shifts) 0-9999 (for index word codes)
P	Digit position	Actual (Enclosed in parentheses)	0-9 (for CD) 0-19 (for split shifts)
S	Unit record synchronizer	Symbolic or actual	1-4
SN	Alteration switch	Symbolic or actual	1-4
SW	Electronic switch	Symbolic or actual	1-30
Q	Inquiry synchronizer	Symbolic or actual	1-2
X	Index Word	Symbolic or actual	1-99

, Used as a separator and must be written on the coding sheet (unless the address which follows is blank).

/ Used to indicate the word "or" (e.g., A/L means either an address or a literal).

Used to indicate an accumulator number (1, 2, or 3, which must appear in place of the # symbol).

Alphabetic List of 7070 Autocoder Imperative Operation Codes

Mnemonic	Operation	Operand	
A#	Add to accumulator #	A/L	F
AA	Add absolute to accumulator 1	A/L	F
AAS#	Add to absolute storage from accumulator #	A	F
AS#	Add to storage from accumulator #	A	F
B	Branch	A	
BAL	Branch if any stacking latch is ON	A	
BAS	Branch if alteration switch is ON	SN,A	
BCB	Branch if channel is busy	C,A	
BCX	Branch compared index word	X,A	
BDL	Branch if disk storage latch is ON	AF,A	
BDX	Branch decremented index word	X,A	
BE	Branch if equal	A	
BES	Branch if electronic switch is ON	SW,A	
BFV	Branch if field overflow	A	
BH	Branch if high	A	
BIX	Branch incremented index word	X,A	
BL	Branch if low	A	
BLX	Branch and load location in index word	X,A	
BM#	Branch if minus in accumulator #	A	
BQL	Branch if inquiry latch is ON	Q,A	
BSC	Branch if sign change	A	
BSF	Branch if electronic switch is ON and set OFF if ON	SW,A	
BSN	Branch if electronic switch is ON and set ON if OFF	SW,A	
BTL	Branch if tape latch is ON	CU,A	
BUL	Branch if unit record latch is ON	I,A	
BV#	Branch if overflow in accumulator #	A	
BXM	Branch if index word is minus	X,A	
BXN	Branch if indexing portion in index word is nonzero	X,A	
BZ#	Branch if zero in accumulator #	A	
C#	Compare accumulator # to storage	A/L	F
CA	Compare absolute in accumulator 1 to absolute in storage	A/L	F
CD	Compare storage to digit	A(P),D	
CSA	Compare sign to alpha	A	
CSM	Compare sign to minus	A	
CSP	Compare sign to plus	A	
D	Divide	A/L	F
DAR	Disk storage arm release	AF	
DLF	Disk storage latch set OFF	AF,A/B	
DLN	Disk storage latch set ON	AF,A/B	
DR	Disk storage read	C,A/L	
DW	Disk storage write	C,A/L	
EAN	Edit alphameric to numerical	X,A/L	
ENA	Edit numerical to alphameric	X,A/L	
ENB	Edit numerical to alphameric with blank insertion	X,A/L	
ENS	Edit numerical to alphameric with sign control	X,A/L	
ESF	Electronic switch set OFF	SW,A/B	
ESN	Electronic switch set ON	SW,A/B	
FA	Floating add	A/L	
FAA	Floating add absolute	A/L	
FAD	Floating add double precision	A/L	

Mnemonic	Operation	Operand	
FADS	Floating add double precision and suppress normalization	A/L	
FBU	Floating branch underflow	A	
FBV	Floating branch overflow	A	
FD	Floating divide	A/L	
FDD	Floating divide double precision	A/L	
FM	Floating multiply	A/L	
FR	Floating round	A/B	
FS	Floating subtract	A/L	
FSA	Floating subtract absolute	A/L	
FZA	Floating zero and add	A/L	
HB	Halt and branch	A	
HMFV	Halt mode for field overflow	A/B	
HMSC	Halt mode for sign change	A/B	
HP	Halt and proceed	A/B	
LE	Lookup equal only	A/L	F
LEH	Lookup equal or high	A/L	F
LL	Lookup lowest	A/L	F
M	Multiply	A/L	F
MSA	Make sign alpha	A	
MSM	Make sign minus	A	
MSP	Make sign plus	A	
NOP	No operation	A/B	
PC	Priority control	A/L	
PDR	Priority disk storage read	C,A/L	
PDS	Priority disk storage seek	A	
PDW	Priority disk storage write	C,A/L	
PR	Priority release	A/B	
PTRA	Priority tape read alpha	CU,A/L	
PTM	Priority tape mark write	CU	
PTR	Priority tape read	CU,A/L	
PTRR	Priority tape read per record mark control	CU,A/L	
PTSB	Priority tape segment backspace	CU,A/L	
PTSF	Priority tape segment forward space	CU,A/L	
PTSM	Priority tape segment mark write	CU	
PTW	Priority tape write	CU,A/L	
PTWC	Priority tape write with zero elimination and per record mark control combined	CU,A/L	
PTWR	Priority tape write per record mark control	CU,A/L	
PTWZ	Priority tape write with zero elimination	CU,A/L	
QLF	Inquiry latch set OFF	Q,A/B	
QLN	Inquiry latch set ON	Q,A/B	
QR	Inquiry read	Q,A/L	
QW	Inquiry write	Q,A/L	
RG	Record gather	X,A/L	
RS	Record scatter	X,A/L	
S#	Subtract from accumulator #	A/L	F
SA	Subtract absolute from accumulator 1	A/L	F
SL	Shift left coupled	N	
SL#	Shift left accumulator #	N	
SLC	Shift left and count coupled	X	
SLC#	Shift left and count accumulator #	X	
SLS	Shift left split	N(P)	
SMFV	Sense mode for field overflow	A/B	

Mnemonic	Operation	Operand	
SMSC	Sense mode for sign change	A/B	
SR	Shift right coupled	N	
SR#	Shift right accumulator #	N	
SRR	Shift right and round coupled	N	
SRR#	Shift right and round accumulator #	N	
SRS	Shift right split	N(P)	
SS#	Subtract accumulator # from storage	A	F
ST#	Store accumulator #	A	F
STD#	Store digits from accumulator # and ignore sign	A	F
TEF	Tape end of file turn OFF	CU	
TLF	Tape latch set OFF	CU,A/B	
TLN	Tape latch set ON	CU,A/B	
TM	Tape mark write	CU	
TR	Tape read	CU,A/L	
TRA	Tape read alpha	CU,A/L	
TRB	Tape record backspace	CU	
TRR	Tape read per record mark control	CU,A/L	
TRU	Tape rewind and unload	CU	
TRW	Tape rewind	CU	
TSB	Tape segment backspace	CU,A/L	
TSEL	Tape select	CU	
TSF	Tape segment forward space	CU,A/L	
TSHD	Tape set high density	CU	
TSK	Tape skip	CU	
TSLD	Tape set low density	CU	
TSM	Tape segment mark write	CU	
TW	Tape write	CU,A/L	
TWC	Tape write with zero elimination and per record mark control combined	CU,A/L	
TWR	Tape write per record mark control	CU,A/L	
TWZ	Tape write with zero elimination	CU,A/L	
TYP	Type	A/L	
ULF	Unit record latch set OFF	I,A/B	
ULN	Unit record latch set ON	I,A/B	
UP	Unit record punch	S,A/L	
UPIV	Unit record punch invalid	S,A/L	
UR	Unit record read	S,A/L	
US	Unit record signal	S,A/B	
UW	Unit record write	S,A/L	
UWIV	Unit record write invalid	S,A/L	
XA	Index word add to indexing portion	X,A/N	
XL	Index word load	X,A/L	
XLIN	Index word load and interchange	X,A/L	
XS	Index word subtract from indexing portion	X,A/N	
XSN	Index word set nonindexing portion	X,A/N	
XU	Index word unload	X,A	
XZA	Index word zero and add to indexing portion	X,A/N	
XZS	Index word zero and subtract from indexing portion	X,A/N	
ZA#	Zero accumulator # and add	A/L	F
ZAA	Zero accumulator 1 and add absolute	A/L	F
ZS#	Zero accumulator # and subtract	A/L	F
ZSA	Zero accumulator 1 and subtract absolute	A/L	F
ZST#	Zero storage and store accumulator #	A	F

Appendix C: *Glossary*

ACTUAL or ACTUAL ADDRESS	The word "actual" usually refers to machine language. An actual address is the same as an absolute or machine address.
ADDRESS ADJUSTMENT	Address adjustment refers to the procedure of changing an address, at process time, according to an increment or decrement placed after the named address, such as NAME1 + 26.
ADDRESS CONSTANT or *ADCON*	An ADCON (address constant) is a constant which is an address rather than a value such as 3.1416 or 1000000000. The characteristic of an ADCON is that the desired numerical value is usually not known when the program is being written, but is inserted by Autocoder at process time.
ALPHAMERIC	Refers to characters which may be numerical digits, alphabetic letters, or special characters.
7070 AUTOCODER	A program which produces a 7070 machine-language object program from a source program written in the 7070 Autocoder language.
7070 BASIC AUTOCODER	The system described in the manual on the 7070 Basic Autocoder, form C28-6078. This is a part of the 7070 Autocoder system, and the 7070 Basic Autocoder language is a part of the 7070 Autocoder language.
COMPILER	There are various types of programming systems. The 7070 Autocoder programming system processor is classed as a compiler. A compiler examines the source language and selects appropriate material from a library, connects it together, and transforms the results into machine language. Compiler is a specific term, as compared with processor, which is the generic term.
EXPRESSION	An element of the source language where a combination of several names and operators may be used, as well as a single name or address.
FIELD DEFINER(S)	A number placed after an address to indicate the particular digit(s) in a word which are occupied by a field.
GENERATOR	A program or sub-program which selects instructions from the library to build machine instructions or other statements described in a source statement. The term "macro generator" is used for the generation process resulting in macro-instructions.
HEADER LINE	A preparatory line in the source language which indicates the function of one or more subsequent detail lines.
INSTRUCTION	Usually means a single entry in machine language or in symbolic machine language, as opposed to a statement, or macro-instruction, which usually means a language entry that can produce many machine-language entries.
LIBRARY	The Autocoder system includes reference material on a tape. This material includes subroutines, macro-instructions and other information. It is referred to as the Library, and the tape is called the Library Tape. The program which creates and updates the tape is called the Librarian.
LITERAL	Literal refers to an entry on the coding sheet of data as distinguished from the location or address of the data.
MACHINE-LANGUAGE CODING	Instructions written in the form which is intelligible to the internal circuitry of the computer. Sometimes called actual or absolute coding.

MACRO-INSTRUCTION	A macro-instruction in the 7070 Autocoder is a line on the coding sheet for which a series of instructions is usually generated or produced. This series of instructions is usually entered in-line or in-sequence, rather than being placed out-of-line and entered by a Branch instruction.
NUMERICAL	In the 7070, numerical usually refers to a field with a plus or minus sign, rather than an alphabetic sign.
OBJECT PROGRAM	The output from a processor. In this case, a 7070 machine-language program assembled from a source program coded in the language described in this bulletin.
OBJECT TIME	The time at which the object program is being run. Opposed to process time, the time at which Autocoder is being run.
OPERAND	Usually means the location or the field being operated on, as distinguished from the operation, which is the *function* taking place. In a machine-language instruction the Operand is the address. The entire field beginning in column 21 on the Autocoder source language card is also called the Operand field.
OPERATION, OPERATION CODE	These terms usually refer to machine functions. Sometimes it is important to distinguish between a mnemonic, e.g., ZA1, and its machine equivalent +13. More often, the context makes it clear whether "operation code" means "mnemonic operation code," or "machine operation code."
OPERATOR	Although the machine operation code could be considered an operator, this term usually refers to such characters as +, −, =, which are said to "operate" on addresses.
PROCESSOR	A program which performs the functions of assembly, compilation, generation, or any similar functions.
PROCESS TIME	The time at which the source program is being changed into an object program by Autocoder. Opposed to object time, the time at which the object program is being run.
SOURCE LANGUAGE	The language in which a problem was coded, e.g., the 7070 Autocoder language.
SOURCE PROGRAM	The original coding of a program, used as input to a processor. Usually refers to a program written in a language other than machine language.
SPECIAL CHARACTER	One of a set of special symbols. Some common special characters are: # $ + = − * ( ) / , . ¤
STATEMENT	Usually a source-language entry on the coding sheet, especially a line which might eventually produce several machine-language instructions, such as the ZSUM statement, or the GET statement.
SUBROUTINE	Usually, a series of instructions entered by a Branch. Opposed to macro-instructions, which are normally entered sequentially, rather than by means of a Branch instruction.
SYMBOL	In Autocoder, symbol is used to refer to a name used instead of a machine address. Thus "symbolic address," "symbolic name," or "symbolic label," conveys that one is not specifying machine addresses.
SYMBOLIC MACHINE LANGUAGE	Usually refers to a language which is similar to machine language except for symbolic addresses and mnemonic operation codes. Autocoder Imperative Operation Codes are examples of instructions in symbolic machine language.

Appendix D: *Note on Optional Characters*

In certain cases, special characters used on IBM printers and other equipment have optional equivalents. In each case the character must be punched according to the card code, regardless of which option has been chosen for printing on the printer in a given installation.

The special characters which have been used in this manual and their optional equivalents for each type wheel configuration available are given in the following table.

Character Used in This Manual	IBM Card Code	Type Wheel Configuration								
		A	B	C	D	E	F	G	H	K
(	0–8–4	%	%	%	%	%	(	%	(	(
) and ¤	12–8–4	¤	¤	¤	¤	<	)	¤	)	)
@	8–4	@	@	@	@	>	–	–	'	@
+	12	&	/	&	–	–	+	+	+	+
/	0-1	1	&	0	/	&	/	/	/	/

Appendix E: Use of the Input/Output Control System

When the IBM 7070 Input/Output Control System is used in source programs to be processed using Four-Tape Autocoder, several restrictions regarding fields, areas, index words, and DA entries must be observed. Except for the restrictions which follow, source programs may use the Input/Output Control System as explained under "Use of the Input/Output Control System with Autocoder" in the 7070 Data Processing System Bulletin "IBM 7070 Input/Output Control System," form J28-6033-1.

When using PUT macro-instructions with Four-Tape Autocoder, the name preceding the word IN must be the name of either an RDW which defines one area or a tape input file. The use of a field name is not allowed unless, of course, the field name is the label of an RDW that defines the field. The name of a card input file may not be used in the PUT macro-instruction if the output file is a tape file. A record from a card input file may be included in a tape output file if the unit record area is defined by one RDW; the PUT would then be written using the name of the RDW, i.e., the name preceding the word IN would be the same as the fourth item in the DUF entry of the card input file.

The index words in the system descriptive entry (DIOCS) must be specified by actual two-digit addresses or omitted, i.e., represented by commas. Symbolic names may be associated with the actual index words through the use of EQU operations which follow the DIOCS entry in the source program sequence; similarly, instructions that follow the DIOCS entry may use the symbolic names IOCSIXF, IOCSIXG, and IOCSIXH as explained in the bulletin describing the Input/Output Control System.

All DTF entries must precede all Input/Output Control System macro-instructions in the source program sequence. Symbolic names may be used in the File Specifications (DTF) to specify index words A and B (Lines 19 and 20 of the DTF) provided the DTF entries appear in the source program sequence ahead of all imperative instructions in the program. The DIOCS entry may appear ahead of the DTF entries, if desired. The symbolic names entered in the DTF for the index words A and B may not be equated to another symbol.

If the DTF entries do not appear in the source program sequence ahead of all imperative instructions which refer to the tape files, index words A and B must be specified in the DTF by actual two-digit addresses. Symbolic names may be associated with the actual index words through use of EQU operations which follow the DTF entries in the source program sequence. Of course, entries for index words A and/or B may remain blank if reference to them is not required.

Comments cards may not be placed between the DTF entries of the files used in the program nor between the subsequent entries under a DTF.

The high assignment counter will be set to the location immediately following the one assigned to the last instruction resulting from the DIOCS entry regardless of the format of any Origin Control statement that may have preceded the DIOCS entry. To avoid unintentional overlap of areas, the setting of the high assignment counter by the DIOCS entry must be considered when writing Origin Control statements that follow the DIOCS entry.

Appendix F: Changes to the Four-Tape Autocoder Manual

The additions, changes and corrections to the original IBM manual (form C28-6102) that are incorporated into this manual are listed below in page number sequence.

Page Number	Description
6	Added an example of symbolic index word in the form xnn.
7	Added a restriction on placement of comments cards.
10	Corrected the size of area to specify 999 words for one record rather than 999 words for one area.
12	Repositioned information on fields that cross word boundaries to associate it with added examples.
13	Added examples of fields that cross word boundaries and methods of referring to data in those fields.
21	Added an example of the second method of assigning two different symbolic names to the same field.
23	Added a caution regarding use of an Origin Control statement of Format 2 after one of Format 3.
23	Added a statement of the effect of a DIOCS entry on the high assignment counter.
24	Added the definition of a Litorigin Segment.
31	Added a method of writing skeleton instructions to allow a choice of parameters.
31	Added a caution regarding the use of actual index words in parameters.
36	Corrected the coding example to save contents of index word 94 before the BLX instruction.
39	Deleted a phrase stating that there is only one pass of Phase 2.
40	Added a statement that the object program tape will have load card indicators in columns 65 and 79.
41	Stated the requirement that a re-assembly run must have at least one addition, change or deletion and that input must be in ascending sequence.
42	Added an example of an ASIGN card.
44	Added the option of placing the number of cards in the operand of an UPDAT card.
44	Added a reference to stacked source programs for the SOURCE-DECK USE TAPE option card.
46	Added a new option for messages.
47	Added references to the tape density option for the LABELSOUT options.
48	Added a reference to the tape density option for the LABELSOUT option.

Page Number	Description
48	Added a comment on labels for stacked output for the LABELS-OUT option.
48	Added a new option for stacked input and/or output.
48	Added items to the list of "permanent" options as supplied on the Systems Tape.
48	Added a comment on the typing of the level number.
48	Changed references to tape units and channels to a combined number in the form CU.
49	Added a reference to the tape density option.
49	Stated the choice of punch alteration switch settings to select the column into which the load card indicator is to be punched.
49	Added a reference to the tape density option.
49	Stated the requirement of at least one correction card for a re-assembly run.
50	Stated the choice of punch alteration switch settings to select the column into which the load card indicator is to be punched.
50	Added a reference to the tape density option.
50	Changed the settings of the card reader alteration switches to ABAA.
51	Added the settings of the console alteration switches.
52	Added the Input/Output Control System halts that may occur during Four-Tape Autocoder processing.
53	Added a new message for System Control and Librarian.
57	Deleted *PGLIN* from the punch error message.
57	Added a new message for Phase 4.
57	Added the Input/Output Control System messages that may occur during Four-Tape Autocoder processing.
67	Added the PTRA instruction to the list.
68	Added the TRA instruction to the list.
71	Added the / character to the table.
72	Added restrictions on the use of the Input/Output Control System with Four-Tape Autocoder.

Index

When more than one page reference for a particular subject is listed, the page number in *italics* indicates the major reference.

$ .....see: Dollar Sign
¤ Character .....see: Lozenge
/ Character .....see: Slash
@ Symbol .....see: Alpha Symbol

Actual Address 3
  EQU 21
  Field Definition 4
  Indexing 5
Adcon 4, *16*
Additions, Re-assembly 45
Address Adjustment 5
  Macro-Instruction Parameter 34
  Skeleton Instruction Operand 34
  ZSUM 30
Address Types 3
Alpha Symbol (@)
  Alphameric Constants 18
  Alphameric Literals 3
  Macro-Instructions 29
  Optional Equivalents 80
Alphameric Constants 18
Alphameric Literals 3
  see also: Literals
Alteration Switches
  Designations 7
  EQU 23
  Stacked Input/Output 53, 58
Area Number in DA 10
ASIGN Card 47
Asterisk
  Comment Card 7
  In Skeleton Routines 35
  Operand Symbol 3, 5, 11, 21, 24
Blank Address 3
Branch Control 25, *26*
CALL 31
Changes
  Library 46
  Permanent Option 46
  Program 48
  Re-assembly 45
Characters, Optional (Appendix D) 80
Coding Sheet 1
Comments in Source Program 7
Console Procedure 57
Constants
  Adcons 16
  Alphameric 18
  DC Header Line 15
  Field Definition of 4
  Literals 3
  Numerical 16
  Origin Control 23
Control Cards, Run 48
Control Operations 23
  Branch Control 26
  End Control 27
  Litorigin Control 25
  Origin Control 23
DA (Define Area) 10
  Area Number 10
  DA Header Line 10
  Record Definition Words 10
  Relative Addressing 11
  Relative Field Definition 14
  Succeeding Entries 12
DA Header Line 10
DC (Define Constant) 15
  Adcons 16
  Alphameric Constants 18
  DC Header Line 15
  Numerical Constants 16
DC Header Line 15
DRDW 20
Declarative Operations 9
Define Area .....see: DA
Define Constant .....see: DC
Define Record Definition Word 20
DELET Cards
  Re-assembly 46
  System Run 47
Deletions
  Source Program 46
  Library 48
Dollar Sign ($) 38
Duplicate Symbol 38
  Macro-Instructions 31
  Subroutine 38
EQU (Equate) 21
  Actual Address 21
  Electronic Switch Number 22
  Index Word 22
  Input/Output Units 23
  Symbolic Address 21
Electronic Switches 7
  EQU 22
Field Definition 4
  Literals 5
  Macro-Instruction Parameter 34
  Relative Field Definition 14
  Numerical Constants 16
  Skeleton Instruction Operand 34
  ZSUM 30
Fields That Bridge Words 13
Flow Charts, Processor (Appendix A) 69
Glossary (Appendix C) 78
Halts and Messages 58

Header Line
DA 10
DC 15
Heading Line 1
Identification 1
Imperative Operations 8
Codes (Appendix B) 74
INCL
In Macro-Instruction 36
In Subroutine 39
Index Words 5
EQU 22
Indexing 5
Macro-Instruction Parameter 34
Other Uses 6
Reservation of 6
Skeleton Instruction Operand 34
ZSUM 30
Input/Output Control System 81
Input/Output Units
Designations 7
EQU 23
INSER Card 47
Instructions, Operating.....see: Operating Instructions
Label 1
Librarian 41
Flow Chart (Appendix A) 69
Messages 58
Library Change Cards 47
Library Changes 46
Library Subroutines .....see: Subroutines
Line 1
Literals 3
Field Definition 4
In Skeleton Routines 36
Litorigin Control 25
Origin Control 23
Litorigin Control 25
Phase 1 43
Litorigin Segment 25
CALL 31
Subroutines 37
Lozenge (¤) 32
Optional Equivalents 80
Macro-Instructions 29
Actual Index Words 34
Blank Characters 29
CALL 31
Commas 29
Duplicate Symbols 31
Input/Output 81
Phase 1 43
Writing of 31
ZSUM 30
Messages 61
Halts and 58
Librarian 61
Phase 1 62
Phase 2 63
Phase 3 64
Phase 4 65
System Control 61
Numerical Constants 16
Numerical Literals 3
Adcon 4, *16*
see also: Literals
Operand 1
Operating Instructions 54
Console Procedure 57
Original Assembly 55
Re-assembly 56
System Run 56
Operation Codes 1
Control 23
Declarative 9
Imperative 8, 74
Options
Cards 49
Changes 46
Optional Characters (Appendix D) 80
Origin Control 23
Index Word Reservation 6
Phase 1 43
Original Assembly Run 46
Control Card 48
Operating Instructions 55
Page Number 1
Parameters, Macro-Instruction 29, 31, 32
Parameters with Slash 33
Permanent Option Changes 46
Phase 1 41
Flow Chart (Appendix A) 70
Halts and Messages 59
Messages 62
Phase 2 .....41, *43*
Flow Chart (Appendix A) 71
Halts and Messages 59
Messages 63
Phase 3 .....41, *44*
Flow Chart (Appendix A) 72
Halts and Messages 59
Messages 64
Phase 4 .....41, *44*
Flow Chart (Appendix A) 73
Halts and Messages 59
Messages 65
Processor 41
Flow Charts (Appendix A) 69
Librarian 41
Organization Schematic 42
Phase 1 41
Phase 2 .....41, *43*
Phase 3 .....41, *44*
Phase 4 .....41, *44*
System Control 41
Program Changes 48
RDW 10
DRDW 20
Re-assembly Run 45
Additions 45

Changes 45
Control Card 49
Deletions 46
Examples 46
Operating Instructions 49
System Control 41
Record Definition Words *10*, 20
Relative Addressing 11
Relative Field Definition 14
Remarks in Source Program 7
Run Control Cards 48
Runs 45
Original Assembly 45
Re-assembly 45
System 46
Skeleton Routine *29*, 31
Of ZSUM Macro 32
Slash (/) 33
Optional Equivalents 80
Stacking 53
Subroutines 37
Avoiding Duplicate Symbols 38
CALL 31
In Macro-Instructions 36
Litorigin Control 25
Origin Control 24
Phase 1 43
Substitution-type Macro-Instructions
see: Macro-Instructions
Succeeding Entries 12
Switches 7
Symbolic Address 3
Address Adjustment 5
EQU 21
Field Definition 4
In Skeleton Routine 32, *34*
Indexing 5
System Control 41
Flow Chart (Appendix A) 69
Messages 61
System Run 46
Control Card 49
Library Changes 46
Operating Instructions 56
Permanent Option Changes 46
Program Changes 48
UPDAT Cards 48
Writing
Macro-Instructions 31
Subroutines 38
ZSUM 30
Skeleton Routine of 32

IBM 7070 Publications

The following IBM 7070 Systems literature has been published as of the date of this manual:

GENERAL INFORMATION MANUALS

Form Number	Title
F22-6517	Introduction to IBM Data Processing Systems
F28-8043	IBM Commercial Translator
F28-8053	The COBOL Translator

REFERENCE MANUALS

Form Number	Title
C28-6078-I	7070 Basic Autocoder
C28-6090	Simulation of the IBM 650 on the IBM 7070
C28-6099	7070 Basic FORTRAN
C28-6102-1	IBM 7070/7074 Four-Tape Autocoder
C28-6110	IBM 7070/7074 Utility Programs
C28-6111	IBM 7070/7074 Generalized Sorting Program: Sort 90

BULLETINS

Form Number	Title
J20-6067	Uses of Index Words in the IBM 7070
J20-8026	Specifications for Submittal and Processing of Social Security Tape Reports
J28-6032-I	IBM 7070 Autocoder
J28-6033-I	IBM 7070 Input/Output Control System
J28-6040	IBM 7070 Sort 90 and Merge 91 Specifications
J28-6041-I	Assembly and Condensing of 7070 Basic Autocoder Programs on the IBM 650
J28-6042-I	Simulation of the IBM 7070 on the IBM 704 and the IBM 7090
J28-6045	IBM 7070 FORTRAN
J28-6047-1	IBM 7070 SPOOL System
J28-6049	IBM 7070 Report Program Generator
J28-6053	Additions to the IBM 7070 Autocoder; Writing Macro Generators for the IBM 7070 Autocoder
J28-6069	IBM 7070 Sort 90 and Merge 91: Timing Estimates; Modifications
J28-6070	Effects of Increased 7070 Tape Capacity and 7070 Tape-Oriented System on IBM Programs
J28-6083	Machine Requirements for IBM Programs and Programming Systems
J28-6095	IBM 7070 Utility Control Panels
J28-6105	IBM 7070/7074 Compiler Systems: Operating Procedure
J28-6112	IBM 7070/7074 Input/Output Control System: Addenda and Errata
J28-8023	Machine Optimal Approximations

The following IBM 7070 Machine literature has been published as of the date of this manual:

GENERAL INFORMATION MANUAL

Form Number	Title
D22-7004-3	7070 Data Processing System

REFERENCE MANUALS

Form Number	Title
A22-6502-1	IBM 700-7000 Series Auxiliary Operations
A22-7003-2	7070 Data Processing System

BULLETINS

Form Number	Title
G22-6545	General Information (7074 Data Processing System)
G22-6562	7072 Data Processing System
G22-6583	Additional Core Storage; Read Binary Tape (7074)
G22-6588	Changes to 7070 Reference Manual

Printed in U.S.A. C28-6102-1

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York